The Bottleneck Complexity of Secure Multiparty Computation

Elette Boyle¹, Abhishek Jain², Manoj Prabhakaran³, and Ching-Hua Yu⁴

¹ IDC Herzliya, elette.boyle@idc.ac.il
² Johns Hopkins University, abhishek@cs.jhu.edu
³ Indian Institute of Technology Bombay, mp@cse.iitb.ac.in
⁴ University of Illinois at Urbana-Champaign, cyu17@illinois.edu

Abstract
In this work, we initiate the study of bottleneck complexity as a new communication efficiency measure for secure multiparty computation (MPC). Roughly, the bottleneck complexity of an MPC protocol is defined as the maximum communication complexity required by any party within the protocol execution.

We observe that even without security, bottleneck communication complexity is an interesting measure of communication complexity for (distributed) functions and propose it as a fundamental area to explore. While achieving $O(n)$ bottleneck complexity (where $n$ is the number of parties) is straightforward, we show that: (1) achieving sublinear bottleneck complexity is not always possible, even when no security is required. (2) On the other hand, several useful classes of functions do have $o(n)$ bottleneck complexity, when no security is required.

Our main positive result is a compiler that transforms any (possibly insecure) efficient protocol with fixed communication-pattern for computing any functionality into a secure MPC protocol while preserving the bottleneck complexity of the underlying protocol (up to security parameter overhead). Given our compiler, an efficient protocol for any function $f$ with sublinear bottleneck complexity can be transformed into an MPC protocol for $f$ with the same bottleneck complexity.

Along the way, we build cryptographic primitives – incremental fully-homomorphic encryption, succinct non-interactive arguments of knowledge with ID-based simulation-extractability property and verifiable protocol execution – that may be of independent interest.


1 Introduction
Secure multi-party computation (MPC) [42, 24] is a fundamental notion in cryptography, enabling a collection of mutually distrusting parties to jointly evaluate a function on their private inputs while revealing nothing beyond the function output. In the past decades, a great deal of research has been dedicated to the design and optimization of efficient MPC protocols.

In this work, we study one fundamental metric of MPC efficiency: the required communication between parties. In particular, we focus on the communication complexity of MPC in large-scale settings, where the number of participants is significant.

In nearly all existing works in MPC literature, the communication complexity goal has been to minimize the total communication of the protocol across all $n$ parties. However, for many important applications, such as peer-to-peer computations between lightweight devices, total costs (such as total communication) are only secondarily indicative of the feasibility of the computation, as opposed to the primary issue of per-party cost. Indeed, while a total communication bound $L$ implies average per-party communication of the protocol is $L/n$, the computation may demand a subset of the parties

¹ For example, optimizing navigation routes based on traffic information contributed by the cell phones of drivers on the road, without revealing the locations of individual users.
to each communicate as much as \( \Theta(L) \). When all parties contribute input to the computation, then \( L \geq n \), meaning these parties must bear communication proportional to the total number of parties. In large-scale distributed settings, or when the protocol participants are lightweight devices, such a requirement could be prohibitive.

**New efficiency measure:** (MPC) Bottleneck Complexity. To address these concerns, we initiate the study of bottleneck complexity of MPC. The bottleneck complexity of a protocol \( \Pi \) is defined as the maximum communication required by any party within the protocol execution. One may further specialize this to incoming versus outgoing communication. The MPC bottleneck complexity of a (distributed) function is the minimum possible bottleneck complexity of a secure MPC protocol for the function. In this work, our goal is to explore this notion as a complexity measure for distributed computations, and to develop secure protocols with low bottleneck complexity.

Bottleneck complexity addresses certain (practically important) aspects ignored by standard communication complexity. For instance, if two messages are transmitted in two different parts of a network, say \( A \rightarrow B \) and \( C \rightarrow D \), they would be delivered faster than two messages sent to/from the same party, say \( A \rightarrow B \) and \( C \rightarrow B \). While both have same total communication, the latter has higher bottleneck communication.

**Bottleneck Complexity without Security.** Before studying bottleneck communication complexity for secure protocols, we first consider this measure for arbitrary protocols without any security considerations. Indeed, this already forms an interesting measure of complexity for (distributed) functions, and we propose it as a fundamental area to explore. As in the case of total communication complexity (which coincides with bottleneck complexity for the case of 2 parties), there is a trivial upper bound of \( O(n) \) bottleneck complexity for any \( n \)-party functionality (with boolean inputs), where all parties simply send their input to a central party who computes the functionality. On the other hand, in many functions, bottleneck complexity brings out structures that total communication complexity overlooks. For instance, in computing say the XOR or AND of \( n \) bits, total communication complexity is \( \Theta(n) \), but the bottleneck complexity is \( O(1) \). These functions naturally allow for incremental computation along a chain, in which each party receives and sends a single bit. Indeed, there is a large class of useful functions which have protocols with low bottleneck complexity, as discussed below.

However, a priori it is not clear whether all functions can be computed in a similar manner. This brings us to the first question considered in this work:

Can all functions be computed (without security)
with sublinear bottleneck complexity?

For concreteness, we may consider \( n \)-party functions, with \( n \) inputs (one for each party) each \( k \) bits long, and a single-bit output. Because of the trivial \( O(nk) \) upper bound on total communication complexity for any such function (as discussed above), each party on average needs only to communicate \( O(k) \) bits. But in this protocol, the communication complexity of the central party—and thus bottleneck complexity of the protocol—is \( (n-1)k \) bits. Surprisingly, we show that this is the best one can ask for, for general functions. That is, there exist \( n \)-party functionalities with \( k \)-bit inputs for which the bottleneck complexity is \( \Omega(nk) \).

**Theorem 1.** (Informal.) There exist \( n \)-party functions with \( k \)-bit input for each party that have bottleneck complexity close to that in the trivial upperbound, namely \( (n-1)k \).

Our proof is based on a counting argument, and quantifies over possibly inefficient functions too. Interestingly, giving an explicit efficient function \( f \) with such a lower bound will require a breakthrough in complexity theory, as it would imply an \( \Omega(n^2) \) lower bound on the circuit size of computing \( f \). (We discuss this connection below.)

**Functions with Low Bottleneck Complexity.** Despite the above lower bound, there is a large class of interesting functions which do have sublinear bottleneck complexity. One simple but widely
applicable example is addition in a finite group: the sum of \( n \) group elements distributed among \( n \) parties can be aggregated bottom-up (and then disseminated top-down) using a constant-degree tree, with every party communicating \( O(d) \) group elements, where \( d \) is its degree in the tree.\(^2\)

A wider class of functions are obtained from the literature on streaming algorithms [22, 31]. Indeed, any streaming algorithm with a small memory and a small number of passes corresponds to a low bottleneck complexity function. (Here, we refer to the actual function that the streaming algorithm computes, which may in fact be an approximation to some other desired function.) This is because we can design a protocol which passes around the state of the streaming algorithm from one party to the next, in the order in which their inputs are to be presented to the algorithm.

On the other hand, low bottleneck complexity protocols appear to be much more general than streaming algorithms. Indeed, observe that the low bottleneck complexity protocol described above has a very special communication structure of a chain (or multi-pass cycle). We leave it as an open problem to separate these two notions – i.e., find functions which have low bottleneck complexity protocols, but do not have low-memory streaming algorithms.

Finally, we note that, any \( n \)-input function with a constant fan-in circuit of subquadratic size (i.e., \( o(n^2) \) gates) has a sublinear bottleneck complexity protocol. To see this, first we note that such a circuit can be made to have constant fan-out as well, by increasing the circuit size by a constant factor.\(^3\) Then, a sublinear bottleneck complexity protocol can be obtained from the circuit by partitioning all the \( o(n^2) \) gates roughly equally among the \( n \) parties, and letting the parties evaluate the gates assigned to them, communicating with each other when wires cross party boundaries. The communication incurred by each party is bounded by the number of wires incident on all the gates it is assigned, which is \( o(n) \).

**MPC Bottleneck complexity.** We next turn our attention to achieving low bottleneck complexity for *secure* computation of functionalities. We focus on the general setting where up to \( n - 1 \) out of \( n \) parties can be corrupted. As a baseline, we observe that the MPC protocol of Dodis *et al.* [19] based on “additive-spooky encryption” can be easily adapted to obtain generic secure computation with \( O(n) \) bottleneck complexity (where \( O(n) \) hides factors of the security parameter). Therefore, as in the insecure setting, we focus on constructing MPC protocols with sublinear \( o(n) \) bottleneck complexity.

Specifically, we ask the question:

If a function \( f \) can be computed with bottleneck complexity \( L \),

can it be computed *securely* with the same bottleneck complexity,

up to a multiplicative overhead of the security parameter?

We note that the goal of sublinear bottleneck complexity is strictly stronger than the recently studied problem of MPC with sublinear communication locality [8]. The locality of a protocol is the maximum number of other parties that any party must communicate with during the course of the protocol. It is easy to see that sublinear bottleneck complexity directly implies sublinear locality (since sending/receiving \( o(n) \) bits means that a party can only communicate with \( o(n) \) neighbors); however, as locality does not place any requirements on the number of bits communicated by a party, the converse is not true. Indeed, without security requirements, every function has an \( O(1) \)-local

---

\(^2\) If the group is not abelian, the tree used should be such that its in-order traversal should result in the parties to be ordered in the same way their inputs are ordered in the sum being computed.

\(^3\) Given a gate with fan-out \( d > 2 \), consider the depth-1 tree \( T \) rooted that gate with \( d \) leaves being the gates to which its outputs are connected. \( T \) can be replaced by an equivalent binary tree \( T' \) with the same root and leaves, and \( d - 2 \) new internal nodes. The new internal nodes of \( T' \) can be “charged” to the leaves of \( T \). On doing this for all gates in the circuit, each gate gets charged at most as many times as its fan-in. Since each gate in the original circuit has constant fan-in, this transformation increases the circuit size by at most a constant factor.
The Bottleneck Complexity of Secure Multiparty Computation

We show a general compiler which transforms any (possibly insecure) efficient multi-party protocol $\Pi$ for computing a function $f$ into a protocol $\Pi'$ for securely computing $f$, preserving the per-party communication and computation requirements up to $O(\lambda^c)$ factors in the security parameter $\lambda$ for small constant $c$. The original protocol $\Pi$ can have an arbitrary communication pattern; however, we require that this pattern must be fixed a priori (independent of inputs) and known to all parties. Our compiler additionally preserves the topology of communication graph of $\Pi$ (and in particular, preserves locality).

**Theorem 2.** (Informal.) There is a transformation which maps any (possibly insecure) efficient protocol with fixed-communication-pattern for an $n$-party distributed function $f$ into a secure MPC protocol for $f$ with asymptotically (as a function of $n$) the same communication and computational requirements per party, and using the same communication graph as the original protocol.

The main tools underlying the result include a new notion of incremental fully homomorphic encryption, which we show can be instantiated from the Learning With Errors (LWE) assumption via [23], as well as zero-knowledge succinct non-interactive arguments of knowledge (ZK-SNARK) [2] with a “ID-based” simulation-extractability property [39, 40]. We rely on a setup that includes a common random string and a (bare) public-key infrastructure, where all the $n$ parties have deposited keys for themselves, and which all the parties can access for free. The setup can be reused for any number of executions.

**Our Contributions.** To summarize, our main contributions in this work are as follows:

- We introduce a new measure of per-party communication complexity for (distributed) functions, called bottleneck complexity.
- We demonstrate the existence of $n$-party functions with $k$ bits of input for each party, that have bottleneck complexity $\Theta(nk)$. Showing an explicit function with $\Omega(n)$ bottleneck complexity will require showing an explicit function with $\Omega(n^2)$ circuit size complexity. On the other hand, we observe that many useful classes of functions do have $o(n)$ bottleneck complexity.
- We show a general transformation from arbitrary efficient protocols to secure MPC protocols (in a model with public setup) that asymptotically (as a function of $n$) preserves the communication and computational requirements per party, and preserves the same communication graph.
- As part of our transformation, we introduce cryptographic primitives—Incremental FHE, Verifiable Protocol Execution—and give a construction of ZK-SNARKs with an ID-based simulation-extractability property. These may be of independent interest.

We expand on the transformation in the following section.

**1.1 Our Techniques**

We describe the main ideas underlying our positive result: the bottleneck-complexity-preserving transformation from arbitrary protocols to secure ones.

At a high-level, we follow an intuitive outline for our compiler: (1) We first compile an insecure protocol into a protocol that is secure against semi-honest (or honest-but-curious) adversaries using fully homomorphic encryption (FHE). (2) We then use zero-knowledge succinct arguments of knowledge (ZK-SNARKs) to compile it into a protocol that is (standalone) secure against malicious adversaries. However, we run into several technical challenges along the way, requiring us to develop stronger guarantees for FHE and SNARKs, as well as some other new ideas. We elaborate on these challenges and our solutions below.

**Semi-honest Security.** A natural starting idea to obtain semi-honest security is to execute an “encrypted” version of the underlying (insecure) protocol by using FHE. Once the parties have the
encrypted output, they execute the FHE decryption process to learn the output. The immediate
problem with implementing this idea in the multiparty setting is which key must we use for encryption
and decryption. If a single party knows the (entire) decryption key, then we cannot guarantee security.

To address this problem, two approaches have been developed in the literature: threshold FHE [1],
where the parties jointly generate a public key for an FHE scheme such that each party only knows a
share of the decryption key, and multi-key FHE [29], where each party has its own public and secret
key pair and FHE evaluation can be performed over ciphertexts computed w.r.t. different public keys.

While these approaches have been shown to suffice for constructing round-efficient MPC protocols,
they are not directly applicable to our setting. This is for two reasons:

- Threshold FHE and multi-key FHE systems are defined in the broadcast model of communication
  where each party gets to see the messages sent by all the other parties. In contrast, our setting is
  inherently point-to-point, where a party only communicates with its neighbors in the communica-
tion graph of the underlying insecure protocol. Indeed, in order to maintain sublinear bottleneck
  complexity, we cannot afford each party to communicate with all the other parties.

- Further, in all known solutions for threshold FHE [1] and multi-key FHE [29, 13, 34, 10, 36], the
  size of one or more protocol messages of each party grows at least linearly with the number of
  parties. This directly violates our sublinear bottleneck complexity requirement.

To address these issues, we define and implement a new notion of incremental FHE (IFHE).
Roughly, an IFHE scheme is defined similarly to threshold FHE, with the following key strengthened
requirements: a “joint” public key can be computed by incrementally combining shares provided
by different parties in an arbitrary order. Similarly, a ciphertext w.r.t. the joint public key can be
decrypted by incrementally combining partial decryption shares provided by parties in an arbitrary
order. Crucially, the intermediate keys and partial decryption values must be succinct.

We construct an IFHE scheme with appropriate security guarantees based on the Gentry-Sahai-
Waters FHE scheme [23]. Using IFHE, we are able to directly compile an insecure protocol into a
semi-honest secure protocol. In fact, this protocol can withstand a slightly stronger adversary – called
a semi-malicious adversary [1] – which is allowed to maliciously choose its random tape. This will be
crucially exploited in the next step, because without it, one will need to enforce honest random-tapes
for all the parties (using n-way coin-tossing-in-the-well) which would incur \( \Omega(n) \) communication
already.

From Semi-Malicious to Malicious Security. A natural approach to achieve security against mali-
cious adversaries is to use the GMW paradigm [24]. Roughly, in the GMW compiler, each party first
commits to its input and random tape. Later, whenever a party sends a message of a semi-malicious
protocol, it also proves in ZK to all the other parties that it is behaving correctly w.r.t. the committed
input and random tape.

The GMW commit-and-prove methodology is problematic in our setting since we cannot allow
a party to talk to all other parties (directly or indirectly through the other nodes). Yet, in order to
achieve security, each honest party must verify not just that its neighbors behave correctly, but that all
corrupt parties (many of whom may not directly interact with any honest party) behaved honestly. A
priori, these may seem to be contradictory goals.

We address all of these challenges by presenting a new generic compiler for Verifiable Protocol
Execution (VPE), modeled as a functionality \( F_{vpe} \). Our protocol \( \Pi_{vpe} \) for implementing \( F_{vpe} \) asymp-
totically preserves the per-party communication and computational complexity (up to a multiplicative
factor polynomial in the security parameter) of the underlying semi-malicious protocol. We construct

---

4 The standard GMW compiler is defined for semi-honest protocols and also involves a coin-tossing step. Here, we
consider a natural variant that works for semi-malicious protocols.
Π_{vpe} from two main ingredients: (1) a new commitment protocol that allows the parties to compute a succinct “aggregate” commitment over the inputs and randomness of all of the parties. (2) ZK-SNARKs with a strong extraction property as well as simulation-soundness to ensure that adversary cannot prove false statements even upon receiving simulated proofs. We refer the reader to technical sections for details on our commitment protocol. Here, we discuss our use of ZK-SNARKs.

**ID-Based Simulation-Extractable ZK-SNARKs.** We rely on ZK-SNARKs to let parties provide not just proofs of correctly computing their own messages, but also of having verified previous proofs recursively. This use of SNARKs for recursive verification resembles prior work on proof-carrying data [12, 3]. The key difference is that proof-carrying data only addresses correctness of computation, whereas in our setting, we are also concerned with privacy. In particular, in order to argue security, we also require these proofs to be simulation-sound with extractability (or simply simulation-extractable), which presents a significant additional challenge.

The core challenge in constructing simulation-extractable ZK-SNARKs (SE-ZK-SNARKs) arises from the inherent limitation that extraction from the adversary must be non-black-box (since the size of the extracted witness is larger than the proof itself), but the adversary receives simulated proofs which he cannot directly produce on his own. Indeed, for this reason SE-ZK-SNARKs are impossible to achieve with strong universal composability (UC) security [27]. To reduce the security of an SE-ZK-SNARK construction to an underlying knowledge assumption (such as standard SNARKs), one must thus either (a) start with an assumption that guarantees non-black-box extraction even in the presence of an oracle (which can be problematic [20]), or (b) somehow in the reduction be able to provide the code to answer the adversary’s simulated proof queries, without voiding the reduction by including the simulation trapdoor itself.

Two recent works have presented constructions of SE-ZK-SNARKs, each adopting a different approach. Groth and Maller [25] embody approach (a), constructing full SE-ZK-SNARKs from a new specific pairing-based knowledge assumption which assumes extraction in the presence of black-box access to an oracle with the trapdoor. Alternatively, Garman et al. [21] take approach (b), basing their construction on standard SNARKs; however, their construction is only applicable to a restricted security model where the statements on which the adversarial prover requests simulated proofs are fixed in advance (in which case these proofs can be hardcoded in the reduction). The case where the adversary’s queries are chosen adaptively as a function of previously simulated proofs (which we need for our transformation) is not currently addressed in this setting.

We provide a new solution for handling adaptive queries, without relying upon oracle-based assumptions as in [25]. We consider an ID-based notion of SE-ZK-SNARK, where each proof is generated with respect to an identity (chosen from a set of identities that are fixed in advance). In our definition, the adversary must fix a set ID* of “honest” identities in advance and can then receive simulated proofs on adaptively chosen statements w.r.t. identities from this set. It must then come up with an accepting proof for a new statement w.r.t. an identity id /∈ ID*.

We show how to transform any SNARK argument system into an ID-based SE-ZK-SNARK by relying on only standard cryptographic assumptions. Very roughly, in our construction, it is possible to “puncture” the trapdoor trap for the CRS w.r.t. an identity set ID*. A punctured trapdoor trapID* can only be used to simulate the proofs w.r.t. identities id ∈ ID*, but cannot be used to simulate proofs w.r.t. identities id /∈ ID*. Using such a punctured trapdoor, we are able to successfully implement approach (b) in the adaptive setting. We implement this idea by using identity-based signatures, which can be readily constructed using certificate chains from a standard signature scheme.

Ultimately, we obtain recursively verifiable ID-based SE-SNARKs generically from signatures and (standard) SNARKs with an “additive extraction overhead.” While the latter is a relatively strong requirement, such primitives have been considered in prior work [14, 3] and appears to be as justified as the standard SNARK assumption.
1.2 Related Work

Communication complexity models. The vast majority of study in communication complexity (c.f. [28]) focuses on the setting of only two parties, in which case the total and bottleneck complexities of protocols align (asymptotically). In the multi-party setting, several models are considered regarding how the input to a function f begins initially distributed among the players. The most common such models are the “number-on-forehead” model, in which parties begin holding all inputs except their own, and the model considered in this work (as is standard in MPC), frequently known as the “number in hand” model, where each party begins with his own input. In all cases, the “communication complexity” within the given model refers to the total communication of all parties.

Communication complexity of MPC. Communication complexity of secure multiparty computation (MPC) has been extensively studied over the years. Communication complexity preserving compilers (c.f. [28]) focuses on the setting of only two parties, in which case the total and bottleneck complexities of, say, f(x, y) are studied. In the setting where the number of parties is larger than the security parameter. For example, Damgård et al. showed that securely evaluating a circuit of size m multiplication gates requires $\tilde{O}(nm)$ total communication in the information-theoretic security setting. This implies a super-linear lower bound for bottleneck complexity. For secure computation, in [17], Damgård et al. showed that securely evaluating a circuit of m multiplication gates requires $\Omega(n^2m)$ total communication in the information-theoretic security setting. This implies a super-linear lower bound for bottleneck complexity in their setting. We note, however, that their lower bound does not apply to us, as we consider computational security, and further, their lower bound does not apply to the setting where the number of parties is larger than the security parameter.

Lower bounds on MPC communication complexity. As discussed, lower bounds on standard multi-party communication complexity cannot directly imply meaningful lower bounds on bottleneck complexity, as no such bound can exceed $\tilde{O}(n)$ (attainable by all parties sending their input to a single party), but this implies only a bound of $\Omega(1)$ bottleneck complexity. For secure computation, in [17], Damgård et al. showed that securely evaluating a circuit of m multiplication gates requires $\Omega(n^2m)$ total communication in the information-theoretic security setting. This implies a super-linear lower bound for bottleneck complexity in their setting. We note, however, that their lower bound does not apply to us, as we consider computational security, and further, their lower bound does not apply to the setting where the number of parties is larger than the security parameter.

2 Preliminaries and Definitions

By $x \in_R \mathbb{Z}_q^n$ we denote that x is uniformly sampled from $\mathbb{Z}_q^n$, and by $x \leftarrow D$ we denote that x is sampled from a distribution D. By $\equiv_c$ we denote computational indistinguishability. We denote an N-party additive secret sharing of x from $\mathbb{Z}_q^n$ by $[x]_q^n$. That is, each $P_i$ owns $x_i \in \mathbb{Z}_q$ such that $x = \sum_{i \in [N]} x_i$. When it is clear, we write [x] for brevity.

Communication Model. We consider a synchronous network among n parties, $P_1, \ldots, P_n$, that allows secure communication between (some) pairs of parties; the channels are authenticated and leak nothing except the number of bits in each message.

An n-party protocol $\pi$ in this model is a “next-message function,” that takes as input the round number $t$, two identities $i, j \in [n]$ and the view of $P_i$, and outputs the message from $P_i$ to $P_j$ in round $t$. The view of a party consists of its input, random-tape, and all the messages received in prior rounds.
The Bottleneck Complexity of Secure Multiparty Computation

We introduce a new per-party communication metric for distributed computations.

Definition 3 (Bottleneck Complexity of Protocol). The individual communication complexity of a party $P_i$ in an $n$-party protocol $\pi$, denoted as $CC_i(\pi)$, is the expected number of bits sent or received by $P_i$ in an execution of $\pi$, with worst-case inputs.

The bottleneck complexity (BC) of an $n$-party protocol $\pi$ is the worst-case communication complexity of any party. That is, $BC(\pi) = \max_{i \in [n]} CC_i(\pi)$.

Definition 4 (Bottleneck Complexity of Function). The bottleneck complexity of an $n$-input function $f$ is the minimum value of $BC(\pi)$ when quantified over all $n$-party distributed protocols $\pi$ which correctly evaluate $f$.

Analogously, we define the MPC bottleneck complexity of $f$ as the minimum $BC(\pi)$ quantified over all $n$-party protocols $\pi$ which securely evaluate $f$.

Admissible Protocols. We will show techniques that transform general (insecure) protocols to secure ones. Here we define the required minimal assumption of the original protocols, which we refer to as admissibility. Roughly, a protocol is admissible if its next-message function is polynomial-time computable and it has a fixed communication pattern.

Below $\mathbb{Z}_+$ denotes the set of non-negative integers.

Definition 5 (Admissible Protocol). Let $f$ be a polynomial function, $k$ be a security parameter, and let $\pi = \{\pi_1, \ldots, \pi_n\}$ be a possibly randomized $n$-party protocol, where $\pi_i$ is a next message function of $P_i$. Let $x = \{x_1, \ldots, x_n\}$ and $r = \{r_1, \ldots, r_n\}$ be the input set and the random string set respectively. Denote $\{m_{i,j}^{t}(x,r)\}_{i,j \in [n]}^{t \in [T]}$ as the set of the messages generated by $\pi(x,r)$, and let $|m_{i,j}^{t}(x,r)| \in [0, f(k)]$ be the length of message from $P_i$ to $P_j$ at time $t$. We say $\pi$ is admissible if it satisfies the following two conditions:

- Polynomial-Time Computable: For each $i$, next-message function $\pi_i$ is expressed by a circuit of fixed polynomial-size in $|x_i| + |r_i|$, with a universally bounded depth.

---

5 While one may argue that it is reasonable to allow zero-cost communication in this manner, it can be abused to communicate large amounts of information at the cost of a single bit, by using a large number of rounds. Further, such signalling cannot be used if multiple such protocols are executed concurrently. Also, practically, the prefix-free communication is more amenable to implementing a synchronous model over an asynchronous network, as delays will not be mistaken for shorter (or empty) messages.

6 Precisely, for each $i \in [n], t \in [T], \{m_{i,j}^{t}(x,r)\}_{j \in [n]}^{t \in [t]} \leftarrow \pi_i \left(x_i, r_i, \{m_{j,i}^{t}(x,r)\}_{j \in [n]}^{t \in [t-1]} \right)$.
- **Fixed Communication Pattern:** A protocol \( \pi \) is said to have a fixed communication pattern if, irrespective of the input and random-tapes of the parties, the total number of rounds \( t_{\text{max}} \) is fixed and there is a function \( \text{len} : [t_{\text{max}}] \times [n] \times [n] \to \mathbb{Z}_+ \) that maps \((t, i, j)\) to the length of message (possibly 0) from \( P_i \) to \( P_j \) at round \( t \) as determined by \( \pi \) for any view of \( P_i \).

Note that above we allow randomized protocols, as some interesting low bottleneck complexity protocols (e.g., those derived from streaming algorithms) tend to be randomized.

## 3 Lowerbound on Bottleneck Complexity of Distributed Functions

In this section we show that for most functions \( f \) on \( n \) inputs (each input could be as short as 1 bit, and the output a single bit delivered to a single party), for any distributed computation protocol \( \pi \) that implements \( f \), the (incoming) bottleneck complexity \( BC(\pi) \) is at least \( n - O(\log n) \) bits. In fact, this holds true even without any security requirement. This is tight in the sense that, even with a security requirement, there is a protocol in which only one party has individual communication complexity \( \Omega(n) \), and all others have communication proportional to their inputs and outputs (with a multiplicative overhead independent of the number of parties).

This is somewhat surprising since many interesting functions do have protocols with constant communication complexity. As mentioned before, any (possibly randomized) function which has a streaming algorithm or a sub-quadratic sized circuit (with small fan-in gates) gives rise to low bottleneck complexity protocols.

To show our lower-bound, we need to therefore rely on functions with roughly a quadratic lower-bound on circuit size. Given the current lack of explicit examples of such functions, we present an existential result, and leave it as a conjecture that there are \( n \)-bit input boolean functions with polynomial sized circuits with bottleneck communication complexity of \( \Omega(n) \). For simplicity, we discuss the case of perfectly correct protocols, but as we shall point out, a small constant probability of error does not change the result significantly. This result says that there is a function (in fact, most functions) such that the best bottleneck complexity is almost achieved by the trivial protocol, in which one party receives the inputs of all the other \( n - 1 \) parties and carries out the computation locally.

- **Theorem 6.** \( \exists f : \{0, 1\}^{k \times n} \to \{0, 1\} \) such that any \( n \)-party, each with \( k \) bits input, distributed computation protocol that computes \( f \) correctly will have at least one party receiving at least \( (n - 1)k - O(\log nk) \) bits in the worst-case.

We provide the proof in Appendix B.

## 4 Incremental FHE

We define and implement a new notion of incremental FHE (IFHE), which is used within our main positive result. We start by providing syntax and security definitions for IFHE in Section 4.1. Next, we recall some preliminaries and the Gentry-Sahai-Waters (GSW) FHE scheme [23] in Appendix A.4. We describe our construction of IFHE in Section 4.2.

### 4.1 Definitions

**(Leveled) Fully Homomorphic Encryption.** A fully homomorphic encryption (FHE) scheme consists of algorithms \((\text{KEYGEN}, \text{ENCRYPT}, \text{EVAL}, \text{DECRYPT})\), where \((\text{KEYGEN}, \text{ENCRYPT}, \text{DECRYPT})\) constitute a semantically secure public-key encryption scheme, and \text{EVAL} refers to the homomorphic evaluation algorithm on ciphertexts. An \( \ell \)-leveled FHE scheme supports homomorphic evaluation of circuits of depth at most \( \ell \).
**Incremental FHE.** An IFHE scheme is defined similarly to threshold FHE [1], with the following key modifications: a "joint" public key can be computed by incrementally combining shares provided by different parties in an arbitrary order. Similarly, a ciphertext w.r.t. the joint public key can be decrypted by incrementally combining partial decryption shares provided by parties in an arbitrary order. Crucially, the intermediate keys and partial decryption values must be succinct.

For technical reasons, it is convenient to describe the joint decryption procedure via three sub-algorithms: A procedure PreDec which pre-processes a homomorphically evaluated ciphertext to be safe for joint decryption; PartDec run by each individual party on a ciphertext (with his share of the secret key) to generate his contribution toward the decryption; and CombineDec which combines the outputs of PartDec from each party for a given ciphertext to reconstruct the final decrypted output. In addition to standard semantic security, we also require the output of PartDec to hide information about the secret key share that was used; this is captured by the Simulatability of Partial Decryption property below.

We proceed to give a formal definition.

**Definition 7 (Incremental FHE).** An incremental fully homomorphic encryption (IFHE) scheme is an FHE scheme with an additional algorithm IFHE.CombineKeys and with Decrypt replaced by three algorithms IFHE.PreDec, IFHE.PartDec and IFHE.CombineDec. By $PK_S$ we denote a combined public key of a subset $S \subseteq [n]$ of parties. Particularly, $PK_{\{i\}} = PK_i$ is generated by $P_i$ using the algorithm KeyGen, and $PK = PK_{\{n\}}$ is the final public key. Similarly, by $v_S$ we denote a combined decryption, and by $v_i$ when $S = \{i\}$. For the completeness of notations, let $PK_S$ and $v_S$ be empty strings when $S = \emptyset$. We describe the syntax of the four algorithms as follows:

- **IFHE.CombineKeys($PK_S, PK_T$):** On input 2 combined public keys $PK_S, PK_T$, where $S \cap T = \emptyset$, output a combined public key $PK_{S \cup T}$.
- **IFHE.PreDec($PK, C$):** On input a final public key $PK$ and a ciphertext $C$, sample a public random $R$, and output a re-randomized ciphertext $C'$ of the same plaintext.
- **IFHE.PartDec($PK, SK_i, C$):** On input a final public key $PK$, $i$th secret key $SK_i$, ciphertext $C$, output a partial decryption $v_i$.
- **IFHE.CombineDec($v_S, v_T$):** On input 2 partial decryptions $v_S, v_T$, where $S \cap T = \emptyset$, if $|S \cup T| < n$, output a partial decryption $v_{S \cup T}$; otherwise, output a plaintext $y$ as the final decryption.

Also, we require the following additional properties:

**Efficiency:** There are polynomials $poly_1(\cdot), poly_2(\cdot)$ such that for any security parameter $\lambda$ and any $S \subseteq [n], S \neq \emptyset$, $|PK_S| = poly_1(\lambda)$ and $|v_S| = poly_2(\lambda)$.

**Correctness:** Given a set of plaintexts and a circuit to evaluate, the correctness of IFHE says that the FHE evaluation of the circuit over the ciphertexts can always be decrypted to the correct value, where the ciphertexts are encryption of plaintexts using a single combined public key.

Furthermore, by “Incremental” FHE, we mean that the final combined public key as well as the final combined decryption can be formed in an arbitrary incremental manner. We defer the formal definition of correctness to Appendix C.

**Semantic security under Combined Keys (against Semi-Malicious Adversary):** Given the parameters prepared in the initial setup, the (corrupted) parties $\{P_j\}_{j \neq i}$, instead of using random strings to compute $\{PK_j, SK_j\}_{j \neq i}$, can use an arbitrary string to generate $\{PK_j, SK_j\}_{j \neq i}$. Then as long as an honest party generates $(PK_i, SK_i)$ independently, the encryption using the final combined public key $(PK_{\{n\}}, SK_{\{n\}})$ is semantically secure.

Formally, consider the following experiment:

1. $(\text{params}) \leftarrow \text{Setup}(1^\lambda, 1^d)$
2. \( \forall j \neq i, \text{Adv computes } (PK_j, SK_j) \) according to \textsc{Keygen}(params) but replaces the randomly sampled string by a chosen one. Then \( \text{Adv} \) computes a combined key \( PK_{[n] \setminus \{i\}} \) according to \textsc{CombineKeys}, picks \( x \in \{0, 1\} \) and sends \( (PK_{[n] \setminus \{i\}}, x) \) to the challenger.

3. The challenger computes \( (PK_i, SK_i) \leftarrow \textsc{Keygen}(\text{params}) \), \( PK \leftarrow \textsc{CombineKeys}(PK_i, PK_{[n] \setminus \{i\}}) \), and chooses a random bit \( \beta \in \{0, 1\} \).
   - If \( \beta = 0 \), it computes \( C = \text{Encrypt}(PK, 0) \).
   - Else, it computes \( C = \text{Encrypt}(PK, x) \).

And it sends \( C \) to \( \text{Adv} \).

4. Finally \( \text{Adv} \) outputs a bit \( \beta' \).

We say that the IFHE scheme has semantic security under combined keys if the advantage \( \Pr[\beta' = \beta] - 1/2 \) is negligible in the security parameter \( \lambda \).

**Simulatability of Partial Decryption:** Let \( x \in \{0, 1\} \) be an input plaintext and \( C' \) be an IFHE encryption of \( x \). There exists a PPT simulator \( \text{Sim} \) which, given the combined public key \( PK \), ciphertext \( C' \), a plaintext output \( y \), an index \( i \in [n] \) and all but the \( i \)-th key \( \{sk_j\}_{j \in [n] \setminus \{i\}} \), produces a simulated partial decryption \( v'_i \) computationally close to the honestly generated value \( v_i \):

\[
\left\{ PK, C', v'_i \right\} \approx \left\{ PK, C', v_i \right\},
\]

where \( v_i \leftarrow \text{IFHE.PartialDec}(PK, SK_i, C') \) and \( v'_i \leftarrow \text{Sim}(PK, \{sk_j\}_{j \in [n] \setminus \{i\}}, y, C') \).

### 4.2 Construction of IFHE

We present an IFHE scheme building on the FHE scheme of Gentry et al. [23]. The \textsc{Setup}, \textsc{Keygen}, \textsc{Encrypt}, and \textsc{Eval} parts are the same as those of [23], while \textsc{CombineKeys}, \textsc{PreDec}, \textsc{PartDec} and \textsc{CombineDec} parts are new. The algorithms are described as follows:

#### An IFHE scheme.

- **Setup:** \( (\text{params}) \leftarrow \text{IFHE.Setup}(1^\lambda, 1^d) \). Same as GSW, where \( \text{params} = (q, n, m, \chi, B, \chi, B) \).
- **Key Generation:** \( (PK_i, SK_i) \leftarrow \text{IFHE.Keygen}(\text{params}) \). Same as GSW, where \( PK_i = (B, b_i) \) and \( SK_i = t_i \equiv (-s_i, 1) \).
- **Combining Keys:** \( PK_{s\cup T} \leftarrow \text{IFHE.CombineKeys}(PK_S, PK_T) \)
  - On input \( PK_S = (B, b_S) \) and \( PK_T = (B, b_T) \), output \( PK_{\text{S} \cup \text{T}} = (B, b_S + b_T) \). Particularly \( PK_{[n]} \) is abbreviated as \( PK \) or a matrix \( A \).
- **Encryption:** \( C_i \leftarrow \text{IFHE.Encrypt}(PK, x_i) \). Same as GSW.
- **Evaluation:** \( C \leftarrow \text{IFHE.Eval}(C_1, \ldots, C_r; f) \). Same as GSW.
- **Preparing Decryption:** \( C' \leftarrow \text{IFHE.PreDec}(PK_i, C) \)
  - On input \( PK = A \) and \( C \in \mathbb{Z}_q^{n \times m} \), sample a public random matrix \( R \in \{0, 1\}^{m \times m} \) and output \( C' = C + AR \).
- **Partial Decryption:** \( v_i \leftarrow \text{IFHE.PartialDec}(PK_i, SK_i, C') \)
  - On input \( PK = A, \text{sk}_i \equiv t_i \equiv (-s_i, 1) \), and \( C' \in \mathbb{Z}_q^{m \times m} \), sample \( e'_i \leftarrow \chi^m \), set \( t'_i = t_i \) if \( i = 1 \) and \( t'_i = (-s_i, 0) \) if \( i > 1 \), and output \( v_i = (t'_iC' - e'_i)G^{-1}(w^T) \), where \( w = (0, \ldots, 0, |q/2|) \in \mathbb{Z}_q^n \).
- **Combining Decryption:** \( \text{IFHE.CombineDec}(v_S, v_T) \)
  - On input two partial decryptions \( v_S, v_T \) with \( S \cap T = \emptyset \), compute a partial decryption \( v_{S \cup T} = v_S + v_T \). For \( |S \cup T| < n \), output \( v_{S \cup T} \); for \( |S \cup T| = n \), output a plaintext \( y = \left\lfloor \frac{w}{2^T} \right\rfloor \).

In Appendix C.1, we show this scheme fulfills the required properties described in Section 4.1.
The Bottleneck Complexity of Secure Multiparty Computation

Syntax Extension. For ease of use, we extend the syntax of IFHE.JoinKeys for combining several public keys and the syntax of IFHE.CombinedDec for combining several partial decryptions at one time.

- **Combining Keys**: \( \text{PK}_{S_1 \cup \ldots \cup S_t} \leftarrow \text{IFHE.CombineKeys}(\text{PK}_{S_1}, \ldots, \text{PK}_{S_t}) \)
  
  On input \( \text{PK}_{S_1}, \ldots, \text{PK}_{S_t} \), where \( S_1, \ldots, S_t \subseteq [n] \) are disjoint, compute \( \text{PK}_{S_1 \cup S_2}, \text{PK}_{(S_1 \cup S_2) \cup S_3}, \ldots, \text{PK}_{(S_1 \cup \ldots \cup S_{t-1}) \cup S_t} \) incrementally and finally output \( \text{PK}_{S_1 \cup \ldots \cup S_t} \).
  
  (Alternatively, in our scheme \( \text{PK}_{S_1 \cup \ldots \cup S_t} = (B, \sum_{i=1}^{t} b_{S_i}) \).

- **Combining Decryption**: \( v_{S_1 \cup \ldots \cup S_t} \leftarrow \text{IFHE.CombinedDec}(v_{S_1}, \ldots, v_{S_t}) \)
  
  On input partial decryptions \( v_{S_1}, \ldots, v_{S_t} \), where \( S_1, \ldots, S_t \subseteq [n] \) are disjoint, compute a partial decryption \( v_{S_1 \cup S_2}, v_{(S_1 \cup S_2) \cup S_3}, \ldots, v_{(S_1 \cup \ldots \cup S_{t-1}) \cup S_t} \), and finally if \( S_1 \cup \ldots \cup S_t \subseteq [n] \) output \( v_{S_1 \cup \ldots \cup S_t} \); otherwise, a plaintext.
  
  (Alternatively in our scheme \( v_{S_1 \cup \ldots \cup S_t} = \sum_{i=1}^{t} v_{S_i} \), and for \( S_1 \cup \ldots \cup S_t \subseteq [n] \), output \( v_{S_1 \cup \ldots \cup S_t} \); otherwise, \( y = \left\lfloor \frac{w}{q/2} \right\rfloor \).

5 Summary of Further Results

Due to space limitations we defer the remaining technical sections to the appendix.

We begin in Appendix D by demonstrating how to convert an arbitrary admissible multi-party distributed protocol \( \Pi \) (as per Definition 5) for computing a function \( f \) to a protocol \( \Pi_{\text{sm}} \) for computing \( f \) secure against semi-malicious adversaries, while preserving per-party computation and communication. Note that as per Definition 5 the communication pattern of the starting protocol \( \Pi \) can be arbitrary, but we require that it be fixed (i.e., not data dependent).

**Theorem 8.** Let IFHE be an incremental FHE scheme, and \( \Pi \) be an \( n \)-party protocol for evaluating a function \( f \) with fixed communication pattern. Then there exists a protocol \( \Pi_{\text{sm}} \) that securely evaluates \( f \) against up to \( (n-1) \) semi-malicious corruptions, preserving the per-party computation and communication requirements of \( \Pi \) up to \( \text{poly}(\lambda) \) multiplicative factors (independent of the number of parties \( n \)). Moreover, the communication pattern of \( \Pi_{\text{sm}} \) is identical to that of \( \Pi \) plus two additional traversals of a communication spanning tree of \( \Pi \).

At a very high level, the parties will run a one-pass protocol to (incrementally) construct a joint key for the incremental FHE scheme, then execute the original protocol \( \Pi \) underneath FHE encryption, and finally run one more pass to (incrementally) decrypt. See details and proof in Appendix D.

Next, we give a general compiler to transform the above protocol into a maliciously-secure MPC protocol, while preserving the bottleneck complexity. Our compiler relies upon multiple cryptographic ingredients, most notably, ID-based simulation-extractable succinct non-interactive arguments of knowledge (ZK-SNARKs) – a notion that we define and construct in this work (see Appendix E.1).

**Theorem 9.** Let \( \Pi_{\text{sm}} \) be an MPC protocol that securely evaluates a functionality \( f \) against semi-malicious corruptions, as in Theorem 21. Then, assuming the existence of an ID-based simulation-extractable succinct non-interactive arguments of knowledge, non-interactive commitment schemes and a family of collision-resistant hash functions, there exists a compiler that transforms \( \Pi_{\text{sm}} \) into another MPC protocol \( \Pi \) in the (bare) public-key and common reference string model such that \( \Pi \) computes the same functionality \( f \) and preserves the per-party computation and communication of \( \Pi_{\text{sm}} \) up to \( \text{poly}(\lambda) \) multiplicative factors (independent of the number of parties \( n \)).

The construction of this compiler is given in Appendix E. We prove the security of the compiler in Appendix ??.
References


16. Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam D. Smith. Scalable multiparty computation with nearly optimal work and resilience. In Advances in Cryptology


A Additional Background

A.1 Security of MPC

Stand-Alone Security. We use a standard simulation-based security definition for MPC protocols. However, our protocols are not UC-secure (due to their reliance on SNARKs). An appropriate model of simulation-based security in this case is the so-called standalone security model. This is essentially the model of [24], but we present it as a restriction to the UC security definition. Specifically, we define a standalone environment to be one which initiates a single session (of Π of F), and does not communicate with the adversary until all the honest parties terminate. That is, a standalone environment can only interact with the adversary prior to the start of the protocol, and after it terminates.

We say that a protocol Π is a standalone-secure protocol for a functionality F if it UC-securely realizes F (with selective abort) when restricted to standalone environments. Here, in the ideal model, the adversary is allowed to cause individual honest parties to abort, after obtaining the outputs for the corrupt parties. We point out that the definition of UC-security allows a non-black-box simulator that depends on the adversary. (Unlike in UC-security, existence of a simulator in the standalone setting does not imply the existence of a black-box simulator. In UC-security, one may replace the adversary with a dummy adversary which interacts with the actual adversary which is kept inside the
environment; in the standalone setting the dummy adversary cannot interact with the environment during the protocol execution.)

**Semi-Malicious Security for MPC.** Intuitively, a semi-malicious adversary is one who follows the protocol specification (similar to a semi-honest adversary), but who may choose its input and “random” coins for the protocol following any arbitrary PPT strategy. These values may depend (efficiently) on any public setup information such as a CRS or PKI, but must be chosen before the protocol execution begins. Once it has chosen these values, it must follow the protocol as specified, given the chosen input, and using the chosen coins in place of the random coins. We allow the adversary to also abort communication with individual parties at any point in the protocol (which, in our protocols, will invariably result in the honest party aborting).

We remark that a collection of similar but non-identical notions of semi-malicious adversaries have been considered in prior works (e.g., [26, 1]), but with varied requirements on when the adversary must commit to his choice of input/random. We observe that the notion we consider is relatively weak, where all such information is chosen before protocol execution.

More formally, a semi-malicious adversary \(\text{Adv}_i\) is modeled as an interactive Turing machine (ITM) which, in addition to the standard tapes, has a special auxiliary tape. At the start of the protocol, \(\text{Adv}_i\) selects for each corrupted party \(P_i\) an input \(x_i\) and randomness \(r_i\) (which may depend on the original inputs of corrupted parties and public setup information), and writes \(x_i, r_i\) to its special input auxiliary tape. \(\text{Adv}_i\) then honestly follows the protocol specification for the corrupt parties given this input and random tape. At each round, it can also choose any honest party to abort the execution.

We say that a protocol \(\Pi\) evaluating a function \(f\) is semi-malicious secure if it is a standalone secure protocol, but restricted to PPT semi-malicious adversaries.

### A.2 Multisignatures

In a multisignature scheme, a single short object—the multisignature—can take the place of \(n\) signatures by \(n\) signers, all on the same message. The first formal treatment of multisignatures was given by Micali, Ohta, and Reyzin [32]. We consider a variant of the Micali-Ohta-Reyzin model due to Boldyreva [5], as presented in [30]. In this model, the adversary is given a single challenge verification key \(VK\), and a signing oracle for that key. His goal is to output a forged multisignature \(\sigma^*\) on a message \(m^*\) under keys \(VK_1, ..., VK_k\), where at least one of these keys is a challenge verification key (wlog, \(VK_1\)). For the forgery to be nontrivial, the adversary must not have queried the signing oracle at \(m^*\).

For simplicity, we present a slightly weaker version of the security definition achieved by [30], which suffices for our application.

**Definition 10.** A multisignature scheme is a tuple of algorithms

\[
\begin{align*}
\text{KeyGen}(1^k): & \quad \text{Key generation algorithm. Outputs a secret signing key } SK \text{ together with corresponding public verification key } VK. \\
\text{Sign}(SK, m): & \quad \text{Standard signing algorithm, with respect to message } m \text{ and single signing key } SK. \\
\text{Combine}(\{VK_i, \sigma_i\}_{i=1}^k, m): & \quad \text{Takes as input a collection of signatures (or multisignatures) and outputs a combined multisignature, with respect to the union of verification keys.} \\
\text{MultiVer}(\{VK_i\}_{i=1}^k, m, \sigma): & \quad \text{Verifies multisignature } \sigma \text{ with respect to the collection of verification keys } \{VK_i\}_{i=1}^k. \text{ Outputs 0 or 1.}
\end{align*}
\]

\(\text{Note that multisignatures are a special case of aggregate signatures [6], which in contrast allow combining signatures from } n \text{ different parties on } n \text{ different messages.}\)

\(\text{The security game in [30] also allows the adversary the power to choose verification keys on behalf of corrupted parties, as long as he also provides certification that the keys were properly generated.}\)
that satisfies the following properties:

**Correctness:** For any message $m$, any collection of honestly generated signatures $\{\sigma_i \leftarrow \text{Sign}_{sk_i}(m)\}_{i \in I}$ on $m$ (for $I \subset [n]$), the combined multisignature formed by $\bar{\sigma} \leftarrow \text{Combine}(\{vk_i, \sigma_i\}_{i \in I}, m)$ will properly verify with overwhelming probability: $\Pr[1 \leftarrow \text{MultiVer}(\{vk_i\}_{i \in I}, m, \bar{\sigma})] \geq 1 - \negl(k)$.

**Unforgeability:** For any PPT adversary $A$, the probability that the challenger outputs 1 when interacting with $A$ in the following game is negligible in the security parameter $k$:

Setup. The challenger samples $n$ public key-secret key pairs, $(vk_i, sk_i) \leftarrow \text{KeyGen}(1^k)$ for each $i \in [n]$, and gives $A$ all verification keys $\{vk_i\}_{i \in [n]}$. $A$ selects a proper subset $M \subset [n]$ (corresponding to parties to corrupt) and receives the corresponding set of secret signing keys $\{sk_i\}_{i \in M}$.

Signing queries. $A$ may make polynomially many adaptive signature queries, of the form $(m, vk_i)$. For each such query, the challenger responds with a signature $\sigma \leftarrow \text{Sign}_{sk_i}(m)$ on message $m$ with respect to the corresponding signing key $sk_i$.

Output. $A$ outputs a triple $(\bar{\sigma}^*, m^*, \{vk_i\}_{i \in S})$, where $\bar{\sigma}^*$ is an alleged forgery multisignature on message $m^*$ with respect to a subset of verification keys $S \subset [n]$. The challenger outputs 1 if at least one of the provided verification keys $vk_i$ corresponds to a challenge (honest party) key, the message $m^*$ was not queried to the signature oracle with this verification key $vk_i$, and the provided forgery $\sigma^*$ is a valid multisignature: i.e., $1 \leftarrow \text{MultiVer}(\{vk_i\}_{i \in S}, m^*, \sigma^*)$.

The following theorem follows from a combination of the (standard) signature scheme of Waters [41] together with a transformation from this scheme to a multisignature scheme due to Lu et al. [30].

**Theorem 11** ([30]). There exists a secure multisignature scheme with signature size $\text{poly}(k)$ (independent of message length and number of potential signers), based on the Bilinear Computational Diffie-Hellman assumption.

For convenience of notation, we shall use a multisignature scheme also as a normal signature scheme. In that case, we shall write $\text{MS.Verify}(vk_i, m, \sigma)$ instead of $\text{MS.MultiVer}(\{vk_i\}, m, \sigma)$ to indicate that the set of keys involved is singleton.

**Remark.** We note that in our constructions, we can instantiate a multisignature scheme with a simulation-extractable zero-knowledge SNARK with additive overhead (defined below) and standard signatures.

### A.3 Succinct Non-Interactive Arguments of Knowledge

We consider succinct non-interactive arguments of knowledge (SNARKs) with adaptive soundness. Our treatment follows that of Bitansky et al. [2]. We focus attention to publicly verifiable succinct arguments. Due to recent results demonstrating implausibility of SNARKs with respect to arbitrary worst-case auxiliary input (e.g., [4, 9]), we consider a definition parameterized with respect to a particular auxiliary input distribution $Z$.

**Definition 12** ($Z$-auxiliary input SNARK). A triple of algorithms $(\text{crsGen}, \text{Prove}, \text{Verify})$ is a publicly verifiable, adaptively sound succinct non-interactive argument of knowledge (SNARK) for the relation $R$ with respect to auxiliary input distribution $Z$ if the following conditions are satisfied for security parameter $\lambda$:

- **Completeness:** For any $(x, w) \in R$,
  \[
  \Pr[\text{crs} \leftarrow \text{crsGen}(1^\lambda); \pi \leftarrow \text{Prove}(x, w, \text{crs}) : \text{Verify}(x, \pi, \text{crs}) = 1] = 1.
  \]

  In addition, $\text{Prove}(x, w, \text{crs})$ runs in time $\text{poly}(\lambda, |x|, t)$.
Succinctness: The length of the proof π output by Prove(x, w, crs), as well as the running time of Verify(x, π, crs), is bounded by \( p(\lambda, |X|) \), where \( p \) is a universal polynomial that does not depend on \( \mathcal{R} \). In addition, crsGen(\( \lambda \)) runs in time \( \text{poly}(\lambda) \): in particular, crs is of length \( \text{poly}(\lambda) \).

Adaptive Argument of Knowledge: For any non-uniform polynomial-size prover \( P^* \) there exists a polynomial-size extractor \( \mathcal{E}_{P^*} \), such that for all sufficiently large \( \lambda \in \mathbb{N} \) and auxiliary input \( z \leftarrow \mathcal{Z} \), it holds that
\[
\Pr[z \leftarrow \mathcal{Z}; \text{crs} \leftarrow \text{crsGen}(\lambda^\lambda); (x, \pi) \leftarrow \mathcal{E}_{P^*}(z, \text{crs}); (x, \pi, w) \leftarrow \mathcal{E}_{P^*}(z, \text{crs}) : \text{Verify}(x, \pi, \text{crs}) = 1 \land w \notin R(x)] \leq \text{negl}(\lambda).
\]

Extraction with Additive Overhead. We also consider SNARKs where the extractor incurs only an additive overhead in the running time of the adversarial prover. A \( \mathcal{Z} \)-auxiliary-input SNARK is said to satisfy the additive overhead extraction property if there exists a polynomial \( p \) such that for all polynomial time \( P^* \), there exists an \( \mathcal{E}_{P^*} \) as in Definition 12, such that for all \( z \) in the support of \( \mathcal{Z} \) and all \( \text{crs} \) in the support of \( \text{crsGen} \),
\[
\text{RT}(\mathcal{E}_{P^*}(z, \text{crs})) \leq p(\lambda) + \text{RT}(P^*(z, \text{crs})),
\]
where \( \text{RT}(A) \) denotes the running time of an algorithm \( A \).

A.4 GSW FHE Scheme

We follow the notation of [34] throughout this section. We start by recalling some preliminary definitions and then present the FHE scheme of Gentry, Sahai and Waters [23]. We use their FHE scheme as a key building block in our IFHE scheme.

LWE assumption. We first recall the learning with errors assumption [37].

**Definition 13 (LWE Hardness Assumption).** Let \( \lambda \) be a security parameter, \( \chi = \chi(\lambda) \) be a distribution of small values over \( \mathbb{Z} \), \( n = n(\lambda) \) and \( q = q(\lambda) \) be polynomials of \( \lambda \), and \( m = O(n \log q) \). Let \( s \overset{\$}{\leftarrow} \mathbb{Z}_q^{n-1} \), \( B \overset{\$}{\leftarrow} \mathbb{Z}_q^{(n-1) \times m} \), \( e \leftarrow \chi \), and \( b = sB + e \). Then \( (B, b) \overset{\$}{\approx} (B', b') \), where \( (B', b') \overset{\$}{\leftarrow} \mathbb{Z}_q^{n \times m} \).

Public Short Preimage Matrix. We state a useful fact from [33] that is used in the GSW FHE scheme.

**Lemma 14 ([33]).** For any \( m \geq n(\lfloor \log q \rfloor + 1) \), there is a gadget matrix \( G \in \mathbb{Z}_q^{n \times m} \) and an efficient deterministic function \( G^{-1}(\cdot) \) such that for any \( m' \), any \( M \in \mathbb{Z}_q^{n \times m'} \), \( G^{-1}(M) \in \{0, 1\}^{m \times m'} \), and \( GG^{-1}(M) = M \).

In the GSW scheme, the function \( G^{-1}(\cdot) \) is called BitDecomp and multiplication by \( G \) is the BitDecomp\(^{-1} \) operation. For our purposes, we do not need their implementation details.

GSW Construction. We now proceed to describe the GSW FHE scheme.

**Setup:** (params) \( \leftarrow \) GSW.Setup(\( \lambda, 1^d \))
Choose a lattice with dimension parameters \( n = n(\lambda, d) \), \( B_\chi \)-bounded error distribution \( \chi = \chi(\lambda, d) \) and a modulus \( q \) such that LWE\(_{n-1,q,\chi,B_\chi} \) holds. Choose \( m = O(n \log q) \). Finally, choose a random matrix \( B \in \mathbb{Z}_q^{n \times m} \). Output params = \( (q, n, m, \chi, B, B_\chi, B) \).

**Key Generation:** (pk, sk) \( \leftarrow \) GSW.Keygen(params)
We separately describe two sub-algorithms to compute secret-key and public-key respectively:

- GSW.SKGen(params): Sample \( s \overset{\$}{\leftarrow} \mathbb{Z}_q^{n-1} \). Set \( t = (-s, 1) \in \mathbb{Z}_q^n \) and output sk = t.
A Proof. We have:

Let $\text{GSW.pkGen}(\text{params, sk})$: Parse $\text{sk} = (-s, 1) \in \mathbb{Z}_q^n$. Sample $e \leftarrow \chi^m$. Set $b = sB + e \in \mathbb{Z}_q^m$ and $A = \begin{bmatrix} B \\ b \end{bmatrix} \in \mathbb{Z}_q^{n \times m}$. Output $\text{pk} = A$.

- **Encryption:** $C \leftarrow \text{GSW.Encrypt}(\text{pk}, x)$
  On input a message $x \in \{0, 1\}$, choose a short random matrix $R \in \{0, 1\}^{n \times m}$ and compute $C = AR + xG \in \mathbb{Z}_q^{n \times m}$. Output $C$ as the ciphertext.

- **Decryption:** $x' \leftarrow \text{GSW.Decrypt}(\text{sk}, C)$
  On input a ciphertext $C \in \mathbb{Z}_q^{n \times m}$ and secret key $\text{sk} = t$, compute $v = tCG^{-1}(w^T)$, where $w = [0, \ldots, [q/2]] \in \mathbb{Z}^n$. Output $x' = \left\lceil \frac{v}{q/2} \right\rceil$.

On input two ciphertexts $C_1, C_2 \in \mathbb{Z}_q^{n \times m}$, we define homomorphic addition and multiplication:

- GSW.Add($C_1, C_2$): Output $C_1 + C_2 \in \mathbb{Z}_q^{n \times m}$.
- Output the matrix product $C_1G^{-1}(C_2) \in \mathbb{Z}_q^{n \times m}$.

This allows computation of a NAND gate homomorphically by outputting $G = C_1G^{-1}(C_2)$.

The following theorem is proved in [23].

**Theorem 15.** The scheme described above is a secure (leveled) FHE scheme under the LWE_{n-1,q,x,B} assumption.

Note that $tA = e \approx 0$ since $e$ is a small error. For correctness, $v = t(AR + xG)G^{-1}(w^T) \approx xt\frac{w^T}{[q/2]}$ since $R$ and $G^{-1}w^T$ are composed of 0, 1 values. Hence by checking whether $v$ is closer to 0 or $[q/2]$, we can recover $x \in \{0, 1\}$. Let $C_1$ and $C_2$ be encryptions of $x_1$ and $x_2$ respectively. Then, $C^+ = C_1 + C_2$ is such that $tC^+ \approx (x_1 + x_2)tG$ and $C^\times = C_1G^{-1}(C_2)$ is such that $tC^\times \approx (x_1x_2)tG$.

For security, since $A$ is uniformly random over $\mathbb{Z}_q^{n \times m}$, by leftover hash lemma, for a suitable $m = O(n \log q)$, $AR$ is statistically uniform, so as $C = AR + xG$.

**Key Homomorphic Properties of GSW Scheme.** We now show that the GSW scheme satisfies a useful key-homomorphic property, which makes it particularly amenable to convert into a threshold scheme. In particular, we keep the matrix $B$ fixed, then the sum of two key pairs (computed using $B$) gives a new valid key pair.

- **Claim 1.** Let $t_1 = (-s_1, 1)$ and $t_2 = (-s_2, 1)$ be two secret keys. Let $B \in \mathbb{Z}_q^{n-1 \times m}$ be a random matrix and let $e_1$ and $e_2$ be two error vectors. Further, let $A_1 = \begin{bmatrix} B \\ b_1 \end{bmatrix} = \text{GSW.pkGen}(t_1; B; e_1)$ and $A_2 = \begin{bmatrix} B \\ b_2 \end{bmatrix} = \text{GSW.pkGen}(t_2; B; e_2)$. Then, $A = \begin{bmatrix} B \\ b_1 + b_2 \end{bmatrix} = \text{GSW.pkGen}(t_1 + t_2; B; e_1 + e_2)$.

**Proof.** We have: $A_1 = \text{GSW.pkGen}(t_1; B; e_1) = \begin{bmatrix} B \\ s_1B + e_1 \end{bmatrix}$ and $A_2 = \text{GSW.pkGen}(t_2; B; e_2) = \begin{bmatrix} B \\ s_2B + e_2 \end{bmatrix}$. Then,

\[
A = \begin{bmatrix} B \\ b_1 + b_2 \end{bmatrix} = \begin{bmatrix} B \\ (s_1 + s_2)B + (e_1 + e_2) \end{bmatrix} = \text{GSW.pkGen}(t_1 + t_2; B, e_1 + e_2)
\]
B Proof of Theorem 6

Proof. We establish our lower bound using a counting argument. Consider any (possibly randomized) protocol for computing $f$. Since we assume perfect correctness, we can fix the random-tapes to yield the smallest bottle-neck communication complexity, and derive a deterministic protocol.

Consider the “input transcript” of a party to include all the messages it received from all the parties, through out the protocol. Given our requirement that the messages in each link is encoded using prefix-free codes, such an input transcript can be parsed into the sequence of messages $m_{i,j}^t$ received by $P_i$ (for all $i,j$). The same holds for the output transcript for each party.

Now, the behavior of each party $P_i$ is fully specified as a function of its input and its input transcript. That is, the protocol $\pi$ is completely specified by the set of functions $\{\pi_i\}_{i\in[n]}$ each of which maps an input $x_i$ and an input transcript to an output transcripts (not all such functions may correspond to valid protocols, as they may violate causality and let outgoing messages depend on future incoming messages; but all protocols yield such a set of functions).

Suppose the incoming communication for each party for the protocol is $d$ bits. Then the outgoing communication for each party is at most $nd$ bits. Thus, each $\pi_i$ can be written as a function from $\{0,1\}^{d+k}$ to $\{0,1\}^{nd}$. There are $2^{nd+k} \times nd$ such functions. Since a protocol is a combination of $n$ such functions, we have:

$$\text{Number of protocols with bottle-neck at most }d\leq 2^{nd+k} \times nd \times n.$$  

On the other hand, the total number of boolean functions from $nk$ is $2^{nk}$. Hence,

$$2^{nd+k} \times nd \times n \geq 2^{nk} \implies d + k + \log(n^2 d) \geq nk$$

$$\implies d + k + \log(n^3 k) \geq nk$$

$$\implies d \geq (n - 1)k - 3 \log(nk)$$

Thus, if $d \leq nk$, then $d \geq (n - 1)k - 3 \log(nk)$. That is, $d \geq (n - 1)k - O(\log nk)$. \hfill \Box

Remark. If we allow a small constant probability of error, by an averaging argument we can fix the randomness of the parties so that the resulting deterministic protocol will evaluate the function correctly in most of the domain. That is, it evaluates a function whose truth table has a small hamming distance from the original function. The calculation above can be repeated with each protocol accounting for at most $2^{2^{nk}}$ functions for a constant $c < 1$, leading to $d \geq (n - 1)k - O(\log(nk - \log(1/c^2)))$.

C Formal Definition of Correctness for IFHE Scheme

Correctness: Given a set of plaintexts and a circuit to evaluate, the correctness of IFHE says that the FHE evaluation of the circuit over the ciphertexts can always be decrypted to the correct value, where the ciphertexts are encryption of plaintexts using a single combined public key.

Furthermore, by “Incremental” FHE, we mean that the final combined public key as well as the final combined decryption can be form in an arbitrary incremental manner. That is, a $PK_i$ can first combine with any other $PK_j$ to form a combined key $PK_{\{i,j\}}$, and $PK_{\{i,j\}}$ can then combine with any other $PK_k$ to form $PK_{\{i,j,k\}}$ or with $PK_{\{i,j,k\}}$ to form $PK_{\{i,j,k,l\}}$. The final $PK_{\{n\}}$ should work for all possible combining orders as long as it collects $PK_1, ..., PK_n$, and it is similar for the combining decryption. We will use a binary tree to describe the combining order and particularly use $tree^0$ to describe the combining public key and $tree^0$ the combining decryption. For $b = 0$ and 1, $tree^b$ contains $N^b$ nodes $\{S_i^b\}_{i\in[N^b]}$, including a root $S_1^b$. Because the number of leaves equals
n, we have \(N^b = O(n)\). For brevity, we will name a node by its index, and denote the parent of
the \(i\)-th node by parent\((i)\). Also, w.l.o.g., assume each node \(i\) has two children, and particularly
parent\((2) = \text{parent}(3) = 1.\)

**Definition 16** (Correctness). For any sequence of plaintexts \(x_1, \ldots, x_n\) and circuit \(f\) of depth
bounded by \(d\), and for all \(b = 0, 1,\) and for any \(S_1, \ldots, S_{N^n} \in 2^{[n]}\) such that \(S_1^b = [n],\) for all
\(j \in \text{leaves}^b, |S_j^b| = 1,\) and for all \(i, j, k\) with \(i \neq j\) and \(k = \text{parent}(i) = \text{parent}(j),\) \(S_i^b \cap S_j^b = \emptyset\) and
\(S_i^b \cup S_j^b = S_k^b,\) let \(\text{EXP}\).IFHE be the following experiment of an IFHE scheme:

1. \(\text{params} \leftarrow \text{IFHE.SETUP}(1^\lambda, 1^d).\)
2. \(\forall i \in [n], (\text{PK}_i, \text{SK}_i) \leftarrow \text{IFHE.KEYGEN}(\text{params}).\)
3. \(\forall i, j, k \in N^0\) with \(i \neq j\) and \(k = \text{parent}(i) = \text{parent}(j),\)
   \(\text{PK}_{S_i^b} \leftarrow \text{IFHE.COMBINEKEYS}(\text{PK}_{S_i^b}, \text{PK}_{S_j^b}).\)
4. \(\forall i \in [n], C_i \leftarrow \text{IFHE.ENCRYPT}(\text{PK}_i, x_i).\)
5. \(C \leftarrow \text{IFHE.EVAL}(C_1, \ldots, C_n, f).\)
6. \(C' \leftarrow \text{IFHE.PREDEC}(\text{PK}, C).\)
7. \(\forall i \in [n], v_i \leftarrow \text{IFHE.PARTDEC}(\text{PK}, C', \text{SK}_i).\)
8. \(\forall i, j, k \in N^1\) with \(i \neq j\) and \(\text{parent}(i) = \text{parent}(j) = k \neq 1,\)
   \(v_{S_k^b} \leftarrow \text{IFHE.COMBINEDEC}(v_{S_i^b}, v_{S_j^b}).\)
9. Output \(y \leftarrow \text{IFHE.COMBINEDEC}(v_{S_1^b}, v_{S_3^b}).\)

Then the correctness for the scheme holds if and only if
\[
\Pr[\text{EXP}.\text{IFHE}((x_i)_{i \in [n]}; f; \{S_i^b\}_{i \in [N^b]}, b \in \{0, 1\}) = f(x_1, ..., x_n)] = 1.
\]

**C.1 Proof of Security**

**Lemma 17 (Efficiency).** For the IFHE scheme, there are polynomial functions \(\text{poly}_1(\cdot), \text{poly}_2(\cdot)\)
such that for any security parameter \(\lambda\) and any \(S \subseteq [n], S \neq \emptyset, |\text{PK}_S| = \text{poly}_1(\lambda)\) and \(|v_S| = \text{poly}_2(\lambda)\).

*Proof sketch:* This is ascribed to the key homomorphism property as in GSW. Specifically, according
to the scheme, for all \(S, T \subseteq [n], \text{PK}_{S \cup T} = (\text{B}, b_S + b_T),\) where \(b_S + b_T\) is just addition over
\(\mathbb{Z}_q,\) so \(|\text{PK}_{S \cup T}| = |\text{PK}_S| = |\text{PK}_T|\). Similarly, \(v_{S \cup T} = v_S + v_T\) is also addition over \(\mathbb{Z}_q\) so \(|v_{S \cup T}| = |v_S| = |v_T|\). Thus, for all \(S \subseteq [n], S \neq \emptyset, |\text{PK}_S| = |\text{PK}_1| = \text{poly}_1(\lambda)\) and \(|v_S| = |v_1| = \text{poly}_2(\lambda)\)
for some polynomial \(\text{poly}_1\) and \(\text{poly}_2.\)

**Lemma 18 (Correctness).** The IFHE scheme satisfies the correctness. (See Appendix C for a
formal definition of correctness)

*Proof sketch:* Recall that \(\text{A} = \text{PK}_{[n]}, \text{t}_i = \text{SK}_i = (-s_i, 1)\) and \(b_i = s_i \text{B} + e_i.\) Let \(t = (-\sum_{i=1}^n s_i, 1).\) First, for combining keys, because \(\text{PK}_{S \cup T} = (\text{B}, b_S + b_T)\) for all disjoint \(S\) and \(T,\)
we have \(\text{A} = (\text{B}, \sum_{i=1}^n b_i(t_i)).\) Thus, \(\text{tA} = \sum_{i=1}^n e_i \approx 0.\)

Second, suppose for now \(f(x) = x\) so the FHE evaluation is trivial.

Then in the decryption preparation, we re-randomize the ciphertext with a public random \(R\) and
so \(C' = C + AR = AR' + xG\) for some small matrix \(R'.\)

Finally for combining decryption, since \(v_{S \cup T} = v_S + v_T, v_i = (t'_i C + e'_i)G^{-1}(w^T)\) and
\(\sum_{i=1}^n t'_i = (-\sum_{i=1}^n s_i, 1) = \text{t},\) we have:

\[
v_{[n]} + t'_0 CG^{-1}(w^T) = \left(\sum_{i=1}^n t'_i AR' + x(\sum_{i=1}^n t'_i)G + (\sum_{i=1}^n e'_i)\right) G^{-1}(w^T) \approx xtw^T = x[q/2].
\]
The Bottleneck Complexity of Secure Multiparty Computation

Hence by checking whether this value is close to 0 or \([q/2]\), we can recover \(x \in \{0, 1\}\) with probability 1.

For general circuit \(f\), since the encryption is the same of GSW, by a similar demonstration of the homomorphism for GSW, the homomorphism property also holds for the IFHE scheme. Therefore putting them together, we have:

\[
\Pr[\text{EXP}.\text{IFHE}(\{x_i\}_{i \in [\tau]}; f; \{S^{b_i}_{j} \}_{j \in [2^n - 1], b \in \{0, 1\}}) = f(x_1, \ldots, x_\tau)] = 1.
\]

\(\Box\)

[Lemma 19](Semantic Security under Combined Keys). The IFHE scheme is secure under combined keys.

Proof sketch: From KEYGEN, the joint distribution \(\text{PK}_i = (B, b_i)\) is computationally indistinguishable from a uniformly random matrix by the LWE hardness assumption and so is \(\text{PK} = A = (B, b + b')\) for any \(b' = \sum_{j \neq i} b_j\) which is generated independently of \(b_i\). Consequently \(C \leftarrow \text{GSW.Encrypt}(\text{PK}, x) = AR + xG\), where \(R \leftarrow \{0, 1\}^{m \times m}\). By the leftover hash lemma, \(AR\) is computationally indistinguishable from a uniformly random matrix, and so is \(C\) (Otherwise there is a polynomial-time approach to distinguish \((B, b_i)\) from random.)

[Lemma 20](Simulatability of Partial Decryption). For the IFHE scheme, given the output \(y\), the ciphertext \(C'\), all but the \(i\)-th key \(\{\text{SK}_{j_i} \}_{j \in [n] \setminus i}\), the partial decryption \(y\) is simulatable.

Proof sketch: According to the scheme, after \(\text{PREDIC}\), \(C' = C + AR = AR' + yG\) for some small matrix \(R'\). Recall \(\text{PK} = A = (B, b)\), where \(b = \sum_{i=1}^{n} s_i B + e_i\), and \(\text{SK}_i = t_i = (-s_i, 1)\). Let \(t'_0 = (0^{n-1}, 1)\), and recall \(t'_i = (-s_i, 0)\) for \(i > 1\).

First, for \(i > 1\), \(v_i = (t'_i C' - e'_i) G^{-1}(w^T) = -(s_i BR' + e'_i) G^{-1}(w^T)\), and for \(i = 1\), consider \(v_1 - t'_0 C' G^{-1}(w^T) = (t'_1 - t'_0) C' - e'_1) G^{-1}(w^T) = -(s_1 BR' + e'_1) G^{-1}(w^T)\).

Note that the distribution \((BR', v_i) \triangleq (B', b G^{-1}(w^T)) \triangleq (BR', v_1 - t'_0 C' G^{-1}(w^T))\), where \((B', b') \leftarrow \text{Z}_q^{n \times m}\) by the LWE assumption.

Let \(t_{-i} = \left(-\sum_{j \neq i} s_j, 0\right)\) and \(t_{-i} = \left(-\sum_{j \neq i} s_j, 1\right)\) for \(i \neq 1\). Sample \(\bar{e} \leftarrow \chi^m\). Then we construct \(v'_i = y \cdot 2 - (t_{-i} C' + \bar{e}) G^{-1}(w^T)\) for all \(i\).

Note that for \(i \neq 1\), \(v'_i = y - \frac{1}{2} - (s_i BR' + \bar{e} + y t_{-i} G) G^{-1}(w^T)\), and for \(i = 1\), \(v'_1 - t'_0 C' G^{-1}(w^T) = y - \frac{1}{2} - (s_1 BR' + \bar{e} + y t'_0 G) G^{-1}(w^T)\). Since \(y t_{-i} G G^{-1}(w^T) = y t'_0 G G^{-1}(w^T)\), we have \(v'_i = -(s_i BR' + \bar{e}) G^{-1}(w^T)\) for \(i \neq 1\) and \(v'_1 - t'_0 C' G^{-1}(w^T) = -(s_1 BR' + \bar{e}) G^{-1}(w^T)\). By LWE assumption, the distribution

\[
(BR', v'_i) \overset{c}{=} (B'^{i'}, b'^{i'} G^{-1}(w^T)) \overset{c}{=} (BR', v'_1 - t'_0 C' G^{-1}(w^T)),
\]

where \((B'^{i'}, b'^{i'}) \leftarrow \text{Z}_q^{n \times m}\).

Thus, for all \(i\), \(v_i \overset{c}{=} v'_i\), and

\[
\mathbb{E} \left[ \frac{(v_i + \sum_{j \neq i} t'_i C' G^{-1}(w^T))^2}{q/2} \right] = y = \mathbb{E} \left[ \frac{(v'_i + \sum_{j \neq i} t'_i C' G^{-1}(w^T))^2}{q/2} \right],
\]

together implying \(\{A, C', R, v_i\} \overset{c}{=} \{A, C', R, v'_i\}\).

\(\Box\)

**D** Semi-Malicious Protocol

In this section, we demonstrate how to convert an arbitrary admissible multi-party distributed protocol \(\Pi\) (as per Definition 5) for computing a function \(f\) to a protocol \(\Pi_{sm}\) for computing \(f\) secure against semi-malicious adversaries, while preserving the per-party computation and communication
requirements of II up to $\text{poly}(\lambda)$ multiplicative factors, independent of the number of parties $n$. In fact, aside from two additional phases where information is communicated along a spanning tree of the communication network induced by II, our protocol mimics the precise communication patterns of II.

The communication pattern of the starting protocol II can be arbitrary, but we require that it be fixed and known a priori (i.e., not data dependent). The same assumption is made for (a bound on) the message length on each active communication channel in each round. We assume without loss of generality that the output of II is precisely the evaluation of $f$ on parties’ inputs and no additional information.

For simplicity of exposition, we present the transformation for deterministic protocols II; however, as discussed below, our solution can be extended to handle randomized protocols via a simple coin tossing procedure, leveraging the fact that while the semi-malicious adversary can arbitrarily choose his “randomness,” he must commit to these values before the protocol begins.

Let II be a protocol defined by deterministic next-message function with the following syntax:

$$ (\mu_{i,1}, \ldots, \mu_{i,n}) = \text{NextMsg}(i, t, x_i, \text{Transc}(i, r-1)), $$

where: $i$ is the relevant party id, $t$ is the present round number, $x_i$ is party $i$’s secret input (including secret randomness), $\text{Transc}(i, t-1)$ denotes the entire transcript held by party $i$ after the previous round $t-1$, and $(\mu_{i,1}, \ldots, \mu_{i,n})$ denote the respective messages to be sent by party $i$ to respective parties $1, \ldots, n$ in this round (where $\mu_{i,j} = \emptyset$ if no message is to be sent from $i$ to $j$).

Assume a given spanning tree $\text{tree}$ over the underlying network graph induced by II. Let $\text{depth}((\text{tree}) = d$. Denote $P_i$ as the party at the root (level 0), $\text{chldrn}(j)$ be the children set of a party $P_j$, and $\text{parent}(j)$ be the parent of $P_j$ in $\text{tree}$. For $i \in [n]$, let $\text{desc}(i)$ denote the set of all descendants of $i$ in the tree.

The protocol $\Pi_{\text{sm}}$ takes place in three phases, as described below.

### Semi-Malicious Pattern-Preserving Protocol $\Pi_{\text{sm}}$

Let the underlying protocol II be defined by next-message function $\text{NextMsg}$.

**Setup**

1. $\text{params} \leftarrow \text{IFHE.\text{Setup}(1^\lambda, 1^d)}$
2. All parties receive $\text{params}$ as a common random string.

**Phase 1: Key setup**

1. For $\ell = d, \ldots, 0$: For every $i \in [n]$ for which $P_i$ is at level $\ell$ of $\text{tree}$,
   a. Aggregate public keys:
      i. Denote the received public keys (if any) as $\{PK_{\text{desc}(j)}\}_{j \in \text{chldrn}(i)}$.
      ii. Generate a IFHE key pair: $(PK_i, SK_i) \leftarrow \text{IFHE.\text{KEYGEN}(\text{params})}$.
      iii. Combine keys: $PK_{\text{desc}(i)} \leftarrow \text{IFHE.\text{COMBINEKEYS}(PK_i, \{PK_{\text{desc}(j)}\}_{j \in \text{chldrn}(i)})}$.
   b. Aggregate randomness values (used to rerandomize output ciphertext):
      i. Denote the received random strings (if any) as $\{\text{rand}_j\}_{j \in \text{chldrn}(i)}$.
      ii. Sample a random string: $\text{rand}_i \leftarrow \{0, 1\}^\lambda$.
      iii. Combine random values: $\text{rand}_{\text{desc}(i)} = \bigoplus_{j \in \text{chldrn}(i)} \text{rand}_j$.
   c. If $\ell = 0$ (i.e., root node), let $PK := PK_{\text{desc}(i)}$ and $r := \text{rand}_{\text{desc}(i)}$.
   d. Else, if $\ell \neq 0$, send $PK_{\text{desc}(i)}$ to parent node $\text{parent}(i)$.
2. For $\ell = 0, \ldots, d-1$: For every $i \in [n]$ for which $P_i$ is at level $\ell$ of $\text{tree}$,
   a. Let $PK$ be the key received from parent $\text{parent}(i)$. Send $PK$ to all children, $\{P_j | j \in \text{chldrn}(i)\}$.

**Phase 2: Computation** Each party $P_i$ performs the following.

1. Initialize $\text{Transc}(i, 0) \leftarrow \emptyset$.
2. Encrypt input under joint key: $\hat{x}_i \leftarrow \text{IFHE.\text{ENCRYPT}(PK, x_i)}$.
3. For each round $t = 1, \ldots$, rounds of the original protocol, do:
Moreover, the communication pattern of $\Pi$ traversal of the communication spanning tree; Phase 2 directly matches the communication pattern of $\Pi$. The per-party computation and communication in Phases 1 and 3 are each $\text{poly}(\lambda)$. The costs in Phase 2 correspond directly to those of $\Pi$, except that all computation is performed via homomorphic evaluation, and all communication is sent in encrypted form. Thus both metrics have multiplicative overhead $\text{poly}(\lambda)$.

**D.1 Proof of Security of Semi-Malicious MPC Protocol**

In this section, we prove Theorem 21.

The communication pattern of $\Pi_{sm}$ follows by inspection: Phases 1 and 3 each induce a single traversal of the communication spanning tree; Phase 2 directly matches the communication pattern of $\Pi$. The per-party computation and communication in Phases 1 and 3 are each $\text{poly}(\lambda)$. The costs in Phase 2 correspond directly to those of $\Pi$, except that all computation is performed via homomorphic evaluation, and all communication is sent in encrypted form. Thus both metrics have multiplicative overhead $\text{poly}(\lambda)$. 

---

**Theorem 21.** Let IFHE be an incremental FHE scheme, and $\Pi$ be an $n$-party protocol for evaluating a function $f$ with fixed communication pattern. Then the protocol $\Pi_{sm}$ securely evaluates $f$ against semi-malicious corruptions, preserving the per-party computation and communication requirements of $\Pi$ up to $\text{poly}(\lambda)$ multiplicative factors (independent of the number of parties $n$). Moreover, the communication pattern of $\Pi_{sm}$ is identical to that of $\Pi$ plus two additional traversals of a communication spanning tree of $\Pi$. 

**Remark (Handling randomized protocols $\Pi$).** Our transformation can be modified to support randomized protocols $\Pi$ while increasing per-party communication (additively) by only $\text{poly}(\lambda)$, by adding the following “coin tossing” procedure. At the conclusion of the key setup phase, each party $P_i$ samples and encrypts a random $\lambda$-bit string $s_i$ under the joint IFHE key. These values are incrementally aggregated up to the root of the communication spanning tree to a ciphertext $\hat{s}$ of $s := \bigoplus_{i} s_i$, which is then communicated back along the tree to all leaves. In each future round, parties homomorphically evaluate the NextMsg function of $\Pi$, using (encrypted) randomness generated by homomorphically evaluating a pseudo-random function on the (encrypted) seed $\hat{s}$.

We now proceed to prove Theorem 21. The proof takes two main hybrid steps: First, simulating the output computation by relying on the simulatability of partial decryption property of the IFHE. Once this takes place, no knowledge of the honest parties’ secret key shares is required. In the next step, we can thus replace the honest parties’ inputs with encryptions of 0 by relying on the semantic security of the IFHE under combined keys.
We proceed to prove semi-malicious security of $\Pi_{\text{sim}}$. Let $\text{Adv}$ be an arbitrary non-uniform polynomial-time semi-malicious adversary corrupting a set of parties $M \subseteq \{P_1, \ldots, P_n\}$. We demonstrate the existence of an ideal-world simulator $\text{Sim}$ corrupting the same set of parties $M$, such that for any input vector $\vec{x}$, for any auxiliary input $z \in \{0, 1\}^*$, it holds that:

$$\left\{\text{IDEAL}^\ell_{\text{Sim}, M}(1^\lambda, \vec{x}, z)\right\}_{\lambda \in \mathbb{N}} \approx_c \left\{\text{REAL}_{\text{Adv}, M}^\ell(1^\lambda, \vec{x}, z)\right\}_{\lambda \in \mathbb{N}}.$$ 

Consider the following simulator $\text{Sim}(1^\lambda, \{x_i\}_{P_i \in M}, y, z)$, where $y = f(x_1, \ldots, x_n)$ is the output received from the ideal functionality.

1. Setup: $\text{Sim}$ generates honestly executes IFHE setup.
2. Phase 1 (Key Setup): $\text{Sim}$ honestly emulates the actions of honest parties.
3. Phase 2 (Computation): $\text{Sim}$ honestly emulates the actions of honest parties, with the following exception: In Step 2, for each honest party $P_i \notin M$, instead of encrypting the (unknown) input and randomness of $P_i$, $\text{Sim}$ instead generates encryptions $\hat{x}_i, \hat{\text{rand}}_i \leftarrow \text{IFHE.ENCRIPT}(PK, 0)$ of zero.
4. Phase 3 (Decryption): $\text{Sim}$ honestly emulates the actions of honest parties, except that the partial decryption values $\hat{y}_i$ of honest parties are instead generated in the following manner. For all but one honest party $P_i$, evaluate the partial decryption $\hat{y}_i$ of $P_i$ honestly. For $P_{i^*}$, do the following. Let $\hat{y}$ denote the final evaluated ciphertext (known to $\text{Sim}$ since it is sent to all parties). For each corrupt party $P_j$, determine his secret key $SK_j$ by reading off the appropriate region of $P_j$’s committed random tape. Then, $\text{Sim}$ simulates the partial decryption $\hat{y}_{i^*}$ of $P_{i^*}$ (see Definition 7), given the values of $\{y, \hat{y}, \{SK_j\}_{j \in [n] \setminus \{i^*\}}\}$.

We prove indistinguishability of the simulated experiment via the following sequence of hybrids. For each hybrid $\ell \in \{0, 1, 2\}$, denote by $\text{Hyb}_{\text{Adv}, M}^\ell(1^\lambda, \vec{x}, z)$ the distribution on the inputs and outputs of all parties within the Hybrid $\ell$ experiment.

**Hybrid 0:** Real World.

**Hybrid 1:** (Simulatability of partial decryption.) Same as Hybrid 0, except that the partial decryption values $\hat{y}_i$ of honest parties in Phase 3 are instead generated as done by $\text{Sim}$.

- **Claim 2.** For every non-uniform polynomial-time $\text{Adv}$, input vector $\vec{x}$, and auxiliary input $z$, $\text{Hyb}_{\text{Adv}, M}^0(1^\lambda, \vec{x}, z) \cong \text{Hyb}_{\text{Adv}, M}^1(1^\lambda, \vec{x}, z)$.

  **Proof.** The values $\hat{y}_i$ for honest parties $i \neq i^*$ are identically distributed. Note that the evaluated ciphertext $\hat{y}$ is rerandomized by the value $\text{rand} = \bigoplus_{j \in [n]} \text{rand}_j$. Since $\text{Adv}$ is semi-malicious, the choice of $\text{rand}_j$ for corrupt parties $P_j$ were made independently of the honestly sampled $\text{rand}_{i^*}$, meaning that $\text{rand}$ is uniformly distributed. The claim thus follows directly from IFHE simulatability of partial decryption.

- **Hybrid 2:** (Semantic security under combined keys.) Simulated experiment. Namely, same as Hybrid 1, except that the encryptions generated in Step 2 on behalf of honest parties are replaced by encryptions of zero.

- **Claim 3.** For every non-uniform polynomial-time $\text{Adv}$, input vector $\vec{x}$, and auxiliary input $z$, $\text{Hyb}_{\text{Adv}, M}^1(1^\lambda, \vec{x}, z) \cong \text{Hyb}_{\text{Adv}, M}^2(1^\lambda, \vec{x}, z)$.

  **Proof.** Note that already in Hybrid 1 all information about the contribution $SK_{i^*}$ of party $P_{i^*}$ to the joint secret key $SK$ is removed. The claim thus follows directly by IFHE semantic security under combined keys.
From Semi-Malicious to Malicious Security

In this section we describe how to compile a semi-malicious secure protocol into an actively secure protocol, with the same communication graph. Our compiler relies on common reference string (CRS) and a (bare) public-key setup.

Building Blocks. We use the following cryptographic primitives in our compiler:

- A simulation-extractable succinct non-interactive zero-knowledge argument (SE-ZK-SNARK) for a polynomial-time computable relation. Unlike recent works such as [25, 21], our notion of SE-ZK-SNARK is identity-based, where each proof is generated w.r.t. an identity.
- A multisignature scheme (MS.KeyGen, MS.Sign, MS.Combine, MS.MultiVer) for implementing the (bare) public-key setup in our construction. In our setting, we can instantiate a multisignature scheme with a SE-ZK-SNARK with additive overhead together with standard signatures.
- A non-interactive perfectly binding commitment scheme COM.
- A family of collision-resistant hash functions HASHFAMILY.

We refer the reader to Appendix A for the definitions of SNARKs and multisignatures. Below, we present our definition and construction of ID-based SE-ZK-SNARK.

E.1 ID-Based SE-ZK-SNARK

For reasons as discussed in Section 1.1, we consider an ID-based notion of SE-ZK-SNARK, where each proof is generated with respect to an identity (chosen from a set of identities that are fixed in advance). Proofs for multiple statements can be computed w.r.t. the same identity. Crucially, in our definition of simulation-extractability, the adversary must fix a set ID∗ of “honest” identities in advance and can then receive simulated proofs on adaptively chosen statements w.r.t. identities from this set. It must then come up with an accepting proof for a new statement x w.r.t. an identity id /∈ ID∗. We require the existence of a non-black-box extractor who can extract a valid witness for x from such an adversary.

We show how to transform any SNARK argument system into an ID-based SE-ZK-SNARK by relying on only standard cryptographic assumptions. Very roughly, in our construction, it is possible to “puncture” the trapdoor trap for the CRS w.r.t. an identity set ID∗. A punctured trapdoor trapID∗ can only be used to simulate the proofs w.r.t. identities id ∈ ID∗, but cannot be used to simulate proofs w.r.t. identities id /∈ ID∗. Using such a punctured trapdoor, we are able to construct an extractor while still simulating proofs to the adversary. Specifically, in order to extract from an adversarial prover P∗, we consider an augmented code M that consists of the simulator algorithm S with a punctured trapdoor trapID∗ hardwired in its description, together with the code of prover P∗. Any proof requested by the prover P∗ w.r.t. an identity id ∈ ID∗ can be computed by S using the punctured trapdoor. However, since the cheating prover must produce a proof w.r.t. an identity id /∈ ID∗, we can still successfully extract from M by using the SNARK extractor.

Definition. We now present our definition of ID-based SE-ZK-SNARK. Our definition is parameterized w.r.t. an identity set ID. For our application of SE-ZK-SNARK to actively secure MPC, it suffices to work with polynomial-sized identity sets. Below, the prover and verifier algorithms Prove and Verify are extended to receive an identity as an additional input.

Definition 22 (ID-Based SE-ZK-SNARK). A SNARK system (crsGen, Prove, Verify) for a relation R with respect to auxiliary input distribution Z is said to be a SE-ZK-SNARK with respect to identity set ID if there exists a PPT simulator S = (S1, S2, S3) such that the following holds:
Computational Zero-Knowledge: For every \((x, w) \in \mathcal{R}\) and every \(id \in \text{ID}\),
\[
(cr, \text{id}, x, \pi) \approx (cr', \text{id}, x, \pi'),
\]
where \(crs \leftarrow crsGen(1^\lambda), \pi \leftarrow \text{Prove}(crs, x, w, \text{id}), (crs', \text{trap}) \leftarrow S_1(1^\lambda), \text{trap}_id \leftarrow S_2(\text{trap}, \text{id})\) and \(\pi' \leftarrow S_3(crs', \text{id}, \text{trap}_id, x)\).

Simulation-Sound Extractability: There is a negligible function `negl` such that, for any non-uniform PPT prover \(P\), there exists a polynomial-size extractor \(E_P\), such that \(\Pr[\text{out} = 0] \leq \text{negl}(\lambda)\), where the random variable `out` is the output of the following experiment:

1. \((cr, \text{trap}) \leftarrow S_1(1^\lambda)\)
2. \(z \leftarrow \mathcal{Z}\)
3. \(\text{ID}^* \leftarrow P(z, crs) \text{ s.t. } \text{ID}^* \subset \text{ID} \text{ and } |\text{ID}^*| = \text{poly}(\lambda)\).
4. \(\text{trap}_id^* \leftarrow \mathcal{S}_2(\text{trap}, \text{ID}^*)\), for every \(id^* \in \text{ID}^*\).
5. \((id, x, \pi) \leftarrow P(z, crs, \text{trap}_id^*)\), where \(\text{trap}_id^* = \{\text{trap}_id\}_{id^* \in \text{ID}^*}\).
6. \((id, x, \pi, w) \leftarrow E_P(z, crs)\).
7. Output \(0\) if: \(id \notin \text{ID}^* \wedge \text{Verify}(x, \pi, \text{id}) = 1 \wedge (x, w) \notin \mathcal{R}\).

\textbf{Remark.} In our definition of simulation-extractability, we provide the cheating prover \(P\) with a special trapdoor \(\text{trap}_{ID^*}\) that is “punctured” at the identity set \(\text{ID}^*\). Using this special trapdoor, the cheating prover can compute simulated proofs w.r.t. any identity \(id \in \text{ID}^*\) on its own. The correctness of a punctured trapdoor is captured in our definition of computational zero-knowledge.

One could alternatively consider a weaker definition of simulation-extractability where instead of giving a punctured trapdoor to the cheating prover, we only provide him oracle access to the simulator algorithm that uses the trapdoor to simulate proofs w.r.t. identities \(id \in \text{ID}^*\) on statements chosen adaptively by the cheating prover. Clearly, Definition 22 implies this weaker definition. We choose to work with Definition 22 as it enables easier proof of security for our actively secure MPC protocol discussed later in this section.

\textbf{Construction.} We construct an ID-based SE-ZK-SNARK system \((crsGen, \text{Prove}, \text{Verify})\) for any NP language \(L\) and identity set \(\text{ID}\). We will use the following two ingredients: (1) a witness-indistinguishable SNARK system \((crsGen', \text{Prove}', \text{Verify}')\) for an NP language \(L'\) (described below), and (2) an identity-based signature scheme \((\text{Setup}, \text{Keygen}, \text{Sign}, \text{Verify})\) which can be readily constructed using certificate chains from a standard digital signature scheme, which can in turn be based on one-way functions [38].

- \(crsGen(1^\lambda)\): On input a security parameter, compute \(crs' \leftarrow crsGen'(1^\lambda)\) and \((\text{msk}, \text{mvk}) \leftarrow \text{Setup}(1^\lambda)\). Output \(crs = (crs', \text{mvk})\).
- \(\text{Prove}(crs, \text{id}, x, w)\): On input a crs \(crs = (crs', \text{mvk})\), identity \(id\), statement \(x\) and witness \(w\), compute a proof \(\pi' \leftarrow \text{Prove}'(crs', x, w)\) for the statement \(x' = (x, id, \text{mvk})\) s.t. \(x' \in L'\) iff:
  - \(x \in L\), or
  - \(\exists \sigma\) s.t. \(\text{Verify}(\text{mvk}, id, x, \sigma) = 1\).

Here, the prover uses the witness \(w\) to prove the first part of the statement \(x'\). Output \(\pi = \pi'\).

- \(\text{Verify}(crs, id, x, \pi)\): On input a crs \(crs = (crs', \text{mvk})\), identity \(id\), statement \(x\) and proof \(\pi\), compute and output \(\text{Verify}'(crs', x', \pi)\) where \(x' = (x, id, \text{mvk})\).

\textbf{Theorem 23 (ID-Based SE-ZK-SNARK).} Assuming the existence of one-way functions and a witness-indistinguishable SNARK system for NP, the proposed construction is an ID-Based SE-ZK-SNARK for any NP relation.
We start by describing the simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$:

- $\mathcal{S}_1(1^\lambda)$: On input a security parameter, compute:
  - $\text{crs}' \leftarrow \text{crsGen}'(1^\lambda)$
  - $(\text{msk}, \text{mvk}) \leftarrow \text{Setup}(1^\lambda)$
  - Output $(\text{crs} = \text{crs}', \text{mvk})$ and trap $= \text{msk}$.
- $\mathcal{S}_2(\text{trap}, \text{id})$: On input a master trapdoor $\text{trap} = \text{msk}$ and identity id, compute and output a “punctured” trapdoor $\text{trap}_i = \text{sk}_i \leftarrow \text{Keygen}(\text{msk}, \text{id})$.
- $\mathcal{S}_3(\text{crs}, \text{id}, \text{trap}_i, x)$: On input a crs $\text{crs} = (\text{crs}', \text{mvk})$, identity id, trapdoor $\text{trap}_i = \text{sk}_i$ and statement $x$, compute:
  - A signature $\sigma \leftarrow \text{Sign}(\text{sk}_i, x)$ on message $x$ using the signing key $\text{sk}_i$ for identity id.
  - A proof $\pi' \leftarrow \text{Prove}'(\text{crs}', x', \sigma)$ for the statement $x'$ (defined as above) using $\sigma$ as a witness for the second part of $x'$.
  - Output $\pi'$.

Now, note that the crs computed by $\mathcal{S}_1$ is identically distributed to the crs computed by the honest algorithm $\text{crsGen}$. Further, for any identity id and statement $x \in L$, the only difference between a proof computed via $\text{Prove}$ and $\mathcal{S}_2$ is that the former computes a proof for statement $x'$ (defined as above) using a witness for the first part of $x'$, while $\mathcal{S}_3$ uses a witness for the second part of $x'$. However, from the witness indistinguishability property of $(\text{crsGen}', \text{Prove}', \text{Verify}')$, it follows that these proofs are computationally indistinguishable.

We next argue the simulation-sound extractability of our construction. For any cheating prover $P$, our extractor $\mathcal{E}_P$ is the same as defined for the proof system $(\text{crsGen}', \text{Prove}', \text{Verify}')$. From the correctness of $\mathcal{E}_P$, we know that if $P$ outputs an accepting proof $\pi$ for any statement $x$ w.r.t. identity id, then except with negligible probability, $\mathcal{E}_P$ outputs a witness $w^*$ s.t. either $(x, w^*) = (x, w) \in \mathcal{R}$ or $w^* = \sigma$ s.t. $\text{Verify}(\text{mvk}, \text{id}, x, \sigma)$. From the security of identity-based signatures, however, the latter case can only happen with negligible probability.

**Remark.** In our transformation in the next subsection, we will in fact require the proof system for a set of $\mathsf{NP}$ relations $\{R_1, \cdots, R_N\}$, that can be efficiently evaluated given the index of the relation. These relations can be combined into a single “super-relation” $\mathcal{R}$ which consists of $((i, x), w)$ such that $(x, w) \in R_i$, and use a SE-ZK-SNARK system for this relation $\mathcal{R}$. For the sake of readability, the proof and verification for a statement of the form $(i, x)$ with respect to an identity id will be indicated as $\text{Prove}^id_{R_i}(x, w; \text{crs})$ and $\text{Verify}^id_{R_i}(x, \pi, \text{crs})$.

### E.2 Verifiable Protocol Execution

To abstract out the compiler, we present a “Verifiable Protocol Execution” functionality $\mathcal{F}_{\text{vpe}}$ (parametrized by a protocol $\rho$ which is to be verifiably executed) and present a protocol $\Pi_{\text{vpe}}$ (also parametrized by $\rho$) that standalone securely realizes $\mathcal{F}_{\text{vpe}}$ against active corruption. Then, in Appendix E.4 we show that a semi-malicious protocol can be readily turned into a protocol secure against active corruption, using $\Pi_{\text{vpe}}$.

First, for any semi-malicious protocol $\rho$, we define the functionality $\mathcal{F}_{\text{vpe}}(\rho)$ which accepts the inputs for $\rho$ from all the parties; also it accepts the randomness for the corrupt parties from the adversary, and uniformly samples the randomness for the honest parties. Then it executes the protocol $\rho$ honestly using these inputs and randomness. At each round, if no party issues “abort” it sends the messages generated by the execution to the parties to which the messages are addressed. Formally we define $\mathcal{F}_{\text{vpe}}(\rho)$ as follows:
Definition 24 (Functionality $\mathcal{F}_{\text{vpe}}$). The functionality $\mathcal{F}_{\text{vpe}}(\rho)$, parametrized by a semi-malicious protocol $\rho$, is defined as follows:

1. For each $i \in [n]$, if $P_i$ is honest, receive input $x_i$ from $P_i$ and sample a (sufficiently long) bit string $s_i$ uniformly at random; else receive $(x_i, s_i)$ from the adversary. Let $\tilde{x}_i = (x_i, s_i)$.
2. Internally run $\rho$ among virtual parties $\hat{P}_1, \ldots, \hat{P}_n$, with inputs $\tilde{x}_1, \ldots, \tilde{x}_n$ respectively. At each round $\tau$ of this execution carry out the following:
   a. If $P_j$ is corrupt, and in $\rho$, $\hat{P}_i$ sends a message to $\hat{P}_j$, then forward the message to $P_j$.
   b. If the adversary sends (ABORT, $i$), send ABORT to $P_i$, and also terminate $\hat{P}_i$ in the internal execution of $\rho$.
   c. If $\hat{P}_i$ produces an output $y_i$, send $y_i$ to $P_i$.

E.3 Protocol $\Pi_{\text{vpe}}$

For any protocol $\rho$, the protocol $\Pi_{\text{vpe}}(\rho)$ implements $\mathcal{F}_{\text{vpe}}(\rho)$ (stand-alone) securely against active corruption. As described below, $\Pi_{\text{vpe}}(\rho)$ has a setup phase (to create a common reference string), and consists of two phases: an input commitment phase and an execution phase. $\rho$ itself may include a setup phase $\rho_{\text{setup}}$ to create a common reference string. $\rho$ is required to have an a priori determined communication pattern, indicating which parties send messages to which parties in each round (and how many rounds $T$ there are in total). Let $\text{in}(i, \tau)$ denote the set of all $j$ such that at round $\tau$, $\rho$ requires $P_j$ to send a message to $P_i$. We write $\rho(t; (i, j); \{m_{k,i}^\tau\}_{\tau \leq t, k \in \text{in}(i, \tau), \tilde{x}_i})$ to denote the message (possibly empty) that $P_i$ should send to $P_j$ at round $t$, computed from its own input and randomness $\tilde{x}_i$ and the messages $m_{k,i}^\tau$ it received in all previous rounds $\tau < t$. Also, we write $\rho(\text{out}; i; \{m_{k,i}^\tau\}_{\tau \leq t, k \in \text{in}(i, \tau)})$ to indicate the output to be produced by $P_i$ at the end of the protocol.

Setup. The setup contains four parts: the setup of the PKI environment, a hash function $\text{HASH}$ sampled from a hash family $\text{HASHFAMILY}(1^\lambda)$, the common random strings that will be used in the SE-ZK-SNARK system, and the setup of $\rho$ (if any). Formally, $(\text{params}) \leftarrow \Pi_{\text{vpe}}(\rho).\text{Setup}(1^\lambda, 1^d)$, where params are

\[
\{\text{vk}_i, \text{sk}_i\}_{i \in [n]} \leftarrow \text{MS.KeyGen}(1^\lambda),
\text{hash} \leftarrow \text{HASHFAMILY}(1^\lambda),
\text{crs} \leftarrow \text{SNARK.crsGen}(1^\lambda),
\text{params}_\rho \leftarrow \rho.\text{Setup}(1^\lambda, 1^d)
\]

Commitment Phase. The goal of this phase is to construct a short, globally known commitment to the inputs and random-tapes in $\rho$, for all the parties. This phase uses a (fixed, arbitrary) subgraph $T$ of the communication graph of $\rho$, which forms a rooted spanning tree.\(^9\) This phase consists of two passes up and down the tree, starting from the leaves. In the first upward pass, the parties essentially use a Merkle-tree to commit to $\tilde{x}_i = (x_i, s_i)$ for all $i \in [n]$ (using a hash function included in the setup). Along with the hash value, each party also includes a SE-ZK-SNARK proof of correctness in its message to its parent in the tree; correspondingly, each party verifies the the proofs from its children, and also includes this proof in its input to the hash function. Recall that the SE-ZK-SNARK proofs are associated with an identity; all the proofs used in our protocol shall use the prover’s index as the identity. The relation proven to one’s parent includes the fact that the proofs from the children verified. Though the relations are recursively defined, as we shall see, the complexity of the relations will not grow exponentially with depth, as the relation at a node will depend only on the complexity.

\(^9\) We require the communication graph of $\rho$ to be connected and that all parties know this graph (and spanning tree).
of verifying a SE-ZK-SNARK proof for the relations at its children, and not on computing those relations themselves.

This is followed by a downward pass starting from the root, where the final aggregated commitment $\alpha$ is sent to all the nodes. Here again, the messages are accompanied by appropriate proofs. At the end of this pass, the honest parties are assured that the commitment they have received includes their own input and randomness; however, it is possible that the inputs from the other parties are not correctly included.\(^\text{10}\) To address this, we make one more pair of up and down passes, in which a multi-signature of the commitment is created and passed back to all the parties. Verifying this multi-signature assures the honest parties that all the honest parties have received the same commitment.

$T$ is a spanning tree over the set of nodes $[n]$. $\text{chldrn}(j)$ denotes the set of children of $j$, and $\text{parent}(j)$ is the parent of $j$ (other than root) in $T$. W.l.o.g. we assume that the indices of the nodes are sorted such that the root is 1 and for all $j > 1$, $\text{parent}(j) < j$. The party at node $j$ is denoted by $P_j$.

All messages are assumed to contain a header uniquely identifying its role in the protocol (corresponding to the variable name and indices in the description below).

**Commitment Phase of $\Pi_{up}(\phi)$.** The commitment phase uses SE-ZK-SNARKs for sets of NP relations $R^\text{up}_j$ and $R^\text{down}_j$ (with $j$ denoting the prover’s index for the former and the receiver’s for the latter), defined recursively as follows. Here, all the proofs will use the prover’s index as the identity.

- $(\phi, w) \in R^\text{up}_j$, for $j \in \text{leaves}(T)$, iff $\phi = (w, c, r) = \text{COM}(\tilde{x}; r)$, where leaves$(T)$ denotes the set of leaves of $T$.
- $(\phi, w) \in R^\text{up}_j$, for $j \notin \text{leaves}(T)$, iff $\phi = (w, c, r) = \text{COM}(\text{HASH}(c_k)_{k \in \text{chldrn}(j)}, \tilde{x}; r)$ s.t. $c = \text{COM}(\text{HASH}(c_k)_{k \in \text{chldrn}(j)}, \tilde{x}; r)$ and $\forall k \in \text{chldrn}(j)$, \text{SNARK}.Verify$^\text{up}_j((c_k, \beta_k, \text{crs}) = 1$.
- $(\phi, w) \in R^\text{down}_j$, for $j \notin \text{chldrn}(1)$ (i.e., children of root), iff $\phi = (w, c, \alpha, \alpha, \gamma) = \text{COM}(\text{HASH}(c_k)_{k \in \text{chldrn}(1)}, \tilde{x}; r)$ s.t. $c = c_j$, $\alpha = \text{COM}(\text{HASH}(c_k)_{k \in \text{chldrn}(1)}, \tilde{x}; r)$, and $\text{SNARK}.Verify^\text{down}_j((c_k, \beta_k, \text{crs}) = 1, \forall \ell \in \text{chldrn}(1) \setminus \{j\}$.
- $(\phi, w) \in R^\text{down}_j$, for $j \notin \text{chldrn}(1)$, iff $\phi = (w, c, \alpha, w) = \text{COM}(\text{HASH}(c_k)_{k \in \text{chldrn}(1)}, \tilde{x}; r)$ s.t. $\forall k \in \text{chldrn}(1)$, $w = \text{COM}(\text{HASH}(c_k)_{k \in \text{chldrn}(1)}, \tilde{x}; r)$, and $\text{SNARK}.Verify^\text{down}_j((c_k, \beta_k, \text{crs}) = 1, \forall \ell \in \text{chldrn}(1) \setminus \{j\}$.

Below, for the sake of brevity, instead of specifying $\pi \leftarrow \text{SNARK.Prove}^\text{up}_j((c_k, x, w)$, we often leave $w$ implicit and say that $\pi$ is a proof that $x \in L[R]$ with respect to the identity id. (Here $L[R] = \{x \mid \exists w \text{ s.t. } (x, w) \in R\}$.)

1. **Aggregate Commitments with Proofs.**

   a. For $j = n$ to 1, $P_j$ proceeds as follows: If $j$ is not a leaf, first it receives $(c_k, \beta_k)$ from each $k \in \text{chldrn}(j)$ and asserts that $\text{SNARK}.Verify^\text{up}_j((c_k, \beta_k, \text{crs}) = 1$. Then, $P_j$ samples $c_j = \text{COM}(\text{HASH}(c_k)_{k \in \text{chldrn}(j)}, \tilde{x}; r_j)$. Then:
      - $P_j$, for $j > 1$, sends $(c_j, \beta_j)$ to $P_{\text{parent}(j)}$, where $\beta_j$ is a proof that $(c_j, \beta_j) \in L[R^\text{up}_j]$, with associated identity $j$ (prover’s index).
      - $P_1$ sends, $\forall k \in \text{chldrn}(1)$, sends $P_k (\alpha, \sigma_k, \gamma_k)$ where $\alpha = c_1$ where $11 \sigma_k \leftarrow \text{MS.Sign}(\text{sk}_1, c_k)$, and $\gamma_k$ is a proof that $(\text{crs}, c_k, \alpha) \in L[R^\text{down}_k]$, with identity 1.

2. **Distribute the Aggregated Commitment with Proofs.** Above, children of the root leaf already received $\alpha$ from the root, $P_1$.

   a. For $j = 2$ to $n$, $P_j$ receives $(\alpha, \sigma_j, \gamma_j)$ from $P_{\text{parent}(j)}$, and asserts that $\text{MS.verify}(\text{sk}_{\text{parent}(j)}; c_j, \sigma_j) = 1$, and $\text{SNARK}.Verify^\text{down}_j((\phi, \gamma_j, \text{crs}) = 1, \forall \ell = \text{chldrn}(1)$.

---

\(^{10}\) One could have hashed signed inputs to assure each honest party that all honest parties’ inputs are correctly included in the commitment; however, this leaves room for the adversary to supply the honest parties with commitments which have inconsistent inputs for the corrupt parties. The next pass handles both these issues together.

\(^{11}\) Here a normal signature, instead of a multi-signature suffices. We overload MS to keep the setup shorter.
otherwise. If the verification succeeds, it sends \((\alpha, \sigma_k, \gamma_k)\) to \(P_k\) for each \(k \in \text{chldrn}(j)\), where 
\[\sigma_k \leftarrow \text{MS.Sign}(sk_j, c_k),\] 
and \(\gamma_k\) is a proof that \((\text{crs}, \alpha) \in L[R^r_{\ell,\{i\}}]\), with identity \(j\).

3. Create the Multisignature.
   a. For \(j = 2\) to \(n\), \(P_j\) sends a combined multisignature \(\sigma_j = \text{MS.Combine}\{(vk_j, \sigma_{i,t})_{i} \mid i \in \text{chldrn}(j)\} \cup \{vk_j, \sigma_j\}, \alpha\) to parent(\(j\)), where \(\sigma_j = \text{MS.Sign}(sk_j, \alpha)\).
   b. \(P_1\) computes a final combined multisignature

\[\sigma = \text{MS.Combine}\{(vk_1, \sigma_{i,t})_{i} \mid i \in \text{chldrn}(1)\} \cup \{vk_1, \sigma_1\}, \alpha\},\] 
where \(\sigma_1 = \text{MS.Sign}(sk_1, \alpha)\).

4. Verify the Multisignature.
   a. \(\forall k \in \text{chldrn}(1)\), \(P_1\) sends \(\sigma\) to \(P_k\).
   b. For \(j = 2\) to \(n\), \(P_j\) receives \(\sigma\) from \(P_{\text{parent}(j)}\) and checks if \(\text{MS.MultiVer}(\{vk_j\}_{i \in [n]}, \alpha, \sigma) = 1\). If so, it sends \(\sigma\) to \(P_k\), \(\forall k \in \text{chldrn}(j)\); otherwise, \(P_j\) aborts.

**Execution Phase** In each round, each party \(P_i\) sends messages to its outgoing neighbors in the round according the semi-malicious protocol \(\rho\). Meanwhile, similar to the recursive proofs in the commitment phase, \(P_i\) also sends a proof showing that 1) \(P_i\) computes the message honestly using the input committed in the first phase and the messages it receives so far, 2) it computes an aggregated commitment of its input as well as its children’s commits, 3) it verifies the final aggregated commit \(\alpha\) 4) it verifies the proofs that the messages received so far all follow \(\rho\) honestly.

Therefore, by the (additive-overhead) extraction of the SE-ZK-SNARK, the proofs are recursively (computationally) bound to its previous proofs that according to the computation path of the protocol and finally bound to the input. The input is then bound to the final combined commitment that has been multisigned by all the parties in the first phase. Hence we can argue that, unless the collision resistance of the hash function, the soundness of the commitment scheme, the unforgeability of the multisignature scheme, or the extractability of the SE-ZK-SNARK scheme fails, the corrupt parties cannot deviate from the protocol without triggering an abort.

The proofs in the protocol are for relations \(R^{\rho}_{\ell,\{i\}}\) corresponding to proving that a message from \(P_i\) to \(P_j\) in the \(\ell^{th}\) step is correctly computed according to the input and randomness \(\hat{x}_i\), that \(P_i\) committed during the first phase of the protocol and the messages received during the protocol (with proofs, which have been verified by \(P_i\)). This is formalized below.

Recall that \(\text{in}(i, \tau)\) denotes the set of all \(j\) such that at round \(\tau\), \(\rho\) requires \(P_j\) to send a message to \(P_i\). \(R^{\rho}_{\ell,\{i\}}\) is defined as follows:

- \((\phi, w) \in R^{\rho}_{\ell,\{i\}}\) iff \(\phi = (\text{crs}, \alpha, m)\), \(w = (\hat{x}_i, h, r, \gamma, c, \sigma, \{m_{k_i}, \delta_{k_i}\}_{r \in [1..\ell], k \in \text{in}(i,r)})\) such that
  \[m = \rho(\{i; j\}; \{m_{k_i}, \delta_{k_i}\}_{r \in [1..\ell], k \in \text{in}(i,r)}; \hat{x}_i), c = \text{COM}(h, \hat{x}_i), \text{and}\]

\[\text{SNARK}.\text{Verify}_{\rho}((\text{crs}, \alpha, m_{k_i}, \delta_{k_i}, \text{crs}) = 1\]

\(\forall \tau \in [1..\ell], k \in \text{in}(i, \tau)\).

(Note that the last condition is absent for \(\ell = 1\).)

**Execution Phase of \(\Pi_{\text{ops}}(\rho)\).** For each round \(t\), \(P_i\) and \(P_j\), where \(i \in \text{in}(j, t)\):

1. \(P_i\) computes \(m^i_{1,j} = \rho(i; j; \{m_{k_i}, \delta_{k_i}\}_{r \in [1..\ell], k \in \text{in}(i,r)}; \hat{x}_i)\), and a proof \(\delta^i_{1,j}\) that \(m^i_{1,j} \in L[R^r_{\ell,\{i\}}]\) with respect to the identity \(i\). It also computes a signature \(\delta^i_{1,j}\) on \(m^i_{1,j}\) using its signing key \(sk_i\). Then it sends \((m^i_{1,j}, \delta^i_{1,j}, \delta^i_{1,j})\) to \(P_j\).

2. For each \(j \in \text{in}(i, t)\), \(P_i\) receives \((m^i_{1,j}, \delta^i_{1,j})\) from \(P_j\), defines \(\phi = (\text{crs}, \alpha, m^i_{1,j})\), and asserts that

\[\text{SNARK}.\text{Verify}_{\rho}((\phi, \delta^i_{1,j}, \text{crs}) = 1\]

If this is the last round of the protocol, \(P_i\) computes and outputs
\[y_i = \rho(\text{out}, i; \{m_{k_i}, \delta_{k_i}\}_{r \in [1..\ell], k \in \text{in}(i,r)}; \hat{x}_i)\].
We prove the security in Appendix E.5.

E.4 Using VPE To Compile From Semi-Malicious to Malicious Security

If \( \rho \) is a semi-malicious secure protocol for a secure function evaluation functionality \( F \), then there is a simple standalone secure protocol for \( F \) in the the \( \mathcal{F}_{\text{vpe}}(\rho) \)-hybrid model: the honest parties will simply send their inputs for \( F \) to \( \mathcal{F}_{\text{vpe}}(\rho) \), and output the outputs received back (or abort). The security of the protocol follows from the fact that \( \mathcal{F}_{\text{vpe}}(\rho) \) ensures semi-malicious behavior from the adversary in the \( \rho \) execution. Indeed, the only actions allowed for the adversary in \( \mathcal{F}_{\text{vpe}}(\rho) \)—choosing an input and randomness for each corrupt party, followed by aborting communication with an honest party at any point— is allowed by the semi-malicious adversary model.

Further, the above compiler continues to be secure even if we replace \( \mathcal{F}_{\text{vpe}}(\rho) \) with a protocol \( \Pi_{\text{vpe}}(\rho) \) that standalone securely realizes \( \mathcal{F}_{\text{vpe}}(\rho) \). While typically UC security is required for such composition, note that only a single session of \( \mathcal{F}_{\text{vpe}}(\rho) \) is invoked above. Formally, a simulator \( \hat{\text{Sim}}^* \) for the final protocol in the \( F \) ideal model is constructed as follows: Given an adversary \( \text{Adv} \), first we define the (non-black-box) simulator \( \hat{\text{Sim}}_{\text{Adv}} \) for \( \Pi_{\text{vpe}}(\rho) \) to obtain a hybrid execution in the \( \mathcal{F}_{\text{vpe}}(\rho) \), with \( \hat{\text{Sim}} \) as the adversary. Since \( \mathcal{F}_{\text{vpe}}(\rho) \) is internally running \( \rho \) with virtual parties \( P_i \), we “open it up” and combine the execution of \( \{ P_i \}_{i \in C} \) with \( \hat{\text{Sim}}_{\text{Adv}} \) to define a new adversary \( \text{Adv}' \), which is simply a semi-malicious adversary for \( \rho \). Then we use the semi-malicious security of \( \rho \) to obtain the final simulator \( \hat{\text{Sim}}^* \).

E.5 Security of the VPE Protocol

The goal is to show that the environment’s view (which includes the adversary’s view, along with the honest parties’ inputs and outputs) in the real world is indistinguishable from its view in the ideal world of \( \mathcal{F}_f \), when using an appropriately defined simulator.

To prove the security under the active corruption, we define five hybrids, where the first hybrid denotes the real world execution and the last hybrid denotes the simulator in the ideal world. Let \( \hat{x}_i = (x_i, s_i) \) be the input of \( P_i \) and its private random string \( s_i \), and let \( y_i \) be the output of \( P_i \) at the end of the execution. Every message in \( \Pi_{\text{vpe}}(\rho) \) from \( P_i \) to \( P_j \) is of the form \( (\mu_{i,j}^t, \pi_{i,j}^t) \), where \( t \) is the round number, \( \pi_{i,j}^t \) is a SE-ZK-SNARK proof about a statement involving \( \text{crs}, \mu_{i,j}^t \) and possibly \( \alpha \) (which is part of a message \( \mu_{j}^{s} \) for a round \( \tau \) during the commitment phase). The \( \pi \) component is absent in the messages sent during the multi-signature phase of the commitment phase. Here, the \( \mu \) components are computed with no reference to the \( \pi \) components in any of the messages.

For each \( i = 0, \cdots, 4 \), let \( \text{HYBRID}_i \) be the view of the (stand-alone) environment in the experiments summarized below.

1. \( \text{HYBRID}_0 \) is the environment’s view in the execution of \( \Pi_{\text{vpe}}(\rho) \) in the real world. The environment’s view includes the setup, all the messages received by the adversary during the protocol execution, as well as the honest parties’ inputs and outputs \( \{(x_i, y_i)\}_{i \in H} \) from this execution.

2. \( \text{HYBRID}_1 \) is generated by the same experiment as \( \text{HYBRID}_0 \) with the following modification: the simulator \( \{\hat{\text{Sim}}_1, \hat{\text{Sim}}_2, \hat{\text{Sim}}_3\} \) is used to generate the SE-ZK-SNARK scheme’s setup \( \text{crs} \) as well as all the proofs \( \pi_{i,j}^t \) given by the honest parties \( i \in H \).

   It directly follows from the zero-knowledge property of SE-ZK-SNARK that \( \text{HYBRID}_0 \approx \text{HYBRID}_1 \).

3. \( \text{HYBRID}_2 \) is generated by the same experiment as \( \text{HYBRID}_1 \) with the following modification: In the commitment phase, for each \( j \in H \), \( P_j \) uses a commitment to a dummy string as \( c_j \). (Note
that in this hybrid the proofs are already generated by a simulator; the simulator takes $c_j$ as an input, and does not need the message or randomness used to generate $c_j$.

From the hiding property of COM, we have $\text{HYBRID}_1 \approx \text{HYBRID}_2$.

4. $\text{HYBRID}_3$ is in the $\mathcal{F}_\text{vpe}(\rho)$-hybrid model, with a non-blackbox simulator $\text{Sim}$ interacting with the adversary $\text{Adv}$ and the functionality $\mathcal{F}_\text{vpe}(\rho)$. The details of $\text{Sim}$ are shown in Figure 1. Briefly, it behaves as follows:

- $\text{Sim}$ simulates the commitment phase execution of the honest parties in $\text{HYBRID}_2$ faithfully (as it no more depends on their inputs).
- It uses an extractor for SE-ZK-SNARK system to extract the corrupt parties’ inputs and randomness for $\rho$ from the commitment phase, and forwards it to $\mathcal{F}_\text{vpe}(\rho)$.
- It continues with the execution of the phase using the honest parties’ messages received from $\mathcal{F}_\text{vpe}(\rho)$, by adding simulated proofs to them. In this phase $\text{Sim}$ simply verifies all the proofs $\pi_{i,j}$ sent by each corrupt party $P_i$ to any honest party $P_j$, and if any verification fails, it sends (ABORT, $j$) to $\mathcal{F}_\text{vpe}(\rho)$.

To complete the proof, we need to argue that $\text{HYBRID}_2 \approx \text{HYBRID}_3$.

**Extraction from a Tree of Proofs.** We describe an efficient procedure to extract witnesses from a “tree of proofs,” when the proofs are given using a SE-ZK-SNARK system with additive polynomial overhead extraction. Consider a rooted tree $Q$, with an $\mathcal{NP}$ relation $R_u$ associated with each node $u$ of the tree such that $(\phi, w) \in R_u$ only if $\phi = (\text{crs}, x)$ and for all $v \in \text{child}(u)$, Verify$ _R_u(\phi_v, \pi_v, \text{crs}) = 1$ (additional conditions may be included), where $\phi_v = f_v(\phi, w)$ for some efficiently computable functions $f_v, g_v$. Our goal is to extract a consistent set of witnesses $w_v$ for all the nodes $v$ in the tree, from an adversary who gives a proof $\pi_v$ for a statement $\phi_v$, where $v_0$ is the root of the tree, such that (i) if $u = \text{parent}(v)$, then $(\phi_v, w_v) \in R_v$, where, for $v \neq v_0$, $\phi_v = f_v(\phi_u, w_u)$ with $u = \text{parent}(v)$.

We consider an adversary who is given an auxiliary input $z \leftarrow \mathcal{Z}$, a simulated CRS $\text{crs}$, and also a set of trapdoors trap$^*$, for a set of identities $\text{ID}^*$ (for simplicity we consider an a priori fixed set $\text{ID}^*$, which suffices for our purposes). For brevity, we omit identities and trap$^*$ from the description below, with the understanding that all the proofs are with respect to specific identities (associated with the nodes) that are not contained in $\text{ID}^*$. We shall define a set of machines $A_v$, $E_v$ for each node in the tree $Q$ as follows. Each $A_v$ will be defined as a PPT machine that will output a statement-proof pair $(\phi_v, \pi_v)$ for the relation $R_v$, and $E_v$ is the extractor guaranteed to exist by Definition 12, such that $\text{RT}(E_v) \leq \text{RT}(A_v) + p(\lambda)$.

$A_{v_0}$ is the original adversary $\text{Adv}$ which, given inputs $(z, \text{crs})$ and a random tape $r_{v_0}$ outputs $(\phi_{v_0}, \pi_{v_0})$. For all other nodes $v$ in $Q$ we inductively define $A_v$ as follows, in terms of $A_u$ and $E_u$, where $u = \text{parent}(v)$: It takes an input $(z, \text{crs})$ and a random tape $r^u = r^u || r_v$, (length of $r_v$ to be bounded below), such and runs $E_u(z, \text{crs}, r^u; r_v)$ to obtain $(\phi_u, \pi_u, w_u)$, and outputs $(\phi_v, \pi_v) = (f_v(\phi_u, w_u), g_v(\phi_u, w_u))$.

We point out that, thanks to the additive overhead extraction property, $A_v$ remains a polynomial time adversary. Let $q = p + q_0$ be a polynomial where $p$ is as in the definition of additive overhead extraction, and $q_0$ is an upperbound on the time needed to compute $f_v$ and $g_v$ for any node in the tree ($q$ is fixed independent of the adversary). Inductively, suppose that for a node $u$ at a depth $d$, $\text{RT}(A_u) \leq \text{RT}(A_{v_0}) + dq(\lambda)$ (base case being $d = 0$ for $u = v_0$). If $v$ is a child of $u$, then $\text{RT}(A_v) \leq \text{RT}(E_u) + q_0(\lambda) \leq \text{RT}(A_u) + p(\lambda) + q_0(\lambda) \leq \text{RT}(A_{v_0}) + (d+1)q(\lambda)$, by the definition of $A_v$, the additive overhead guarantee, and the inductive assumption and the definition of $q$. Also, w.l.o.g. we keep the length of the random tapes (for $E_u$ and $A_v$) upperbounded by the running time of the machines.
Our final extraction procedure consists of $E_u$ hardwired for all nodes $u$ in $Q$. On input $(z, \text{crs}, r)$ it sets $r^{v_0} = r$ (where $v_0$ is the root of $Q$), samples $r_v$ of the appropriate lengths for each node $v$, defines $r^v$ to be the concatenation of $r_u$ for all ancestors $u$ of $v$ in $Q$, and runs $E_u(z, \text{crs}, r^v; r_v)$ to obtain $w_u$ for each node $u$. If the proof system used is a $Z$-auxiliary input SE-ZK-SNARK for the set of relations $\{R_v\}$ for nodes $v$ in a polynomially large relation-tree $Q$, then by a union bound, the probability that a PPT adversary $\text{Adv}$, given $(z, \text{crs})$ and $\text{trap}_\text{ID}$, as above, outputs $(\phi, \pi)$ that verifies with respect to an $id \not\in \text{ID}^*$, but the above procedure fails to recover a consistent set of witnesses for all nodes is negligible.

$\text{HYBRID}_2 \approx \text{HYBRID}_3$. We briefly sketch the argument that the view of the environment in $\text{HYBRID}_3$ is indistinguishable from that in $\text{HYBRID}_2$. The main idea is to ensure that if the proofs and signatures sent by a corrupt party to an honest party during the execution phase verify, then the accompanying message should match what the functionality $F_{\text{ope}}(\rho)$ would generate and send to the honest party in that round.

To argue this, we shall use the tree extractor from above again on trees of proofs from corrupt parties to honest parties, this time with relations of the form $R_{i,j,t}^\rho$ at the root and consisting of nodes of the form $R_{i',j',t'}^\rho$ and $R_{i,j,t}^{\text{down}}$ (with $i' \in C$, $t' < t$). The extracted witnesses include purported values of $\hat{x}_i$ and set of messages from honest parties (possibly the empty set) that are “connected to” $m_{i,j}^i$ in $\rho$ (as well as (simulated) proofs accompanying those messages). The messages from the honest parties must exactly be the ones that the simulator forwarded from $F_{\text{ope}}(\rho)$, by the unforgeability of the signatures.

We shall also argue that the extracted values $\hat{x}_i$ from the corrupt parties by this extractor must be same ones as extracted by $\text{Sim}$ in the commitment step. This relies on unforgeability of signatures used during the commitment phase.

Recall the forest obtained from $T$ by deleting the honest parties’ nodes. First consider a corrupt party $i'$ that appears in a tree in that forest which is rooted at $i^* \neq 1$. In this case, the value $\hat{x}_{i'}$ extracted (from the commitment phase as well as the execution phase) is related to $c_{i'}$, the aggregated commitment sent by $P_{i'}$ to its parent (which is an honest party). Due to the unforgeability of the signature on $c_{i'}$, in all the extractions, $\hat{x}_{i'}$ will be related to the same value of $c_{i'}$. Further, due to the binding of COM and the collision-resistance of HASH, the value of $\hat{x}_{i'}$ itself should be the same in all extractions.

This does not cover the possibility that if the root is corrupt, it could partition the honest parties into two sets and run independent executions of $\mu$ with them (which is not allowed in the ideal world). It is to prevent this that we use a multi-signature in the commitment phase. Consider a corrupt party $i'$ in a tree $T_1$ in the forest of corrupt parties, where $T_1$ is rooted at 1, the root of $T$ ($T_1$ exists only if 1 itself was corrupt). In this case $\hat{x}_{i'}$ extracted during the commitment phase is bound to $\alpha$. We defined the input extraction (in Figure 1) using an arbitrary honest party $P_k$ such that in $T$, $k$ is adjacent to a leaf of $T_1$. But the unforgeability of the multi-signature ensures that all honest parties must agree on the same $\alpha$. Hence, during the execution phase again, the extracted $\hat{x}_{i'}$ is bound to the same $\alpha$, no matter which honest party is the receiver. As before, due to the binding of COM and collision-resistance of HASH, this ensures that the value of $\hat{x}_{i'}$ itself is the same in all extractions.
Commitment Phase. Sim interacts with the corrupt parties on behalf of the honest parties \( \{ P_i \}_{i \in H} \), according to the HYBRID_3 experiment. During the commitment phase it simply executes the honest parties in HYBRID_2 faithfully as this execution does not depend on their inputs. If any of the proofs \( \beta_i \) (except \( \gamma_j \)) fails to verify during the commitment phase, the corresponding simulated honest party aborts and does not contribute to the multi-signature. In this case Sim will send (ABORT, \( i \)) for all \( i \in H \) after the commitment phase is over, and will stop the simulation.

Input Extraction. If the commitment phase completes with all the proofs \( \beta_i \) and \( \gamma_j \) supplied by the adversary being accepted, Sim will try to extract the inputs \( \hat{x}_i \) for all \( i \in C \), using the tree extractor (described earlier), as follows.

Let \( T \) be the tree used for aggregating the commitments. At the end of the commitment phase, Sim considers the forest obtained by deleting the set of nodes \( H \) from \( T \). Each tree in this forest is denoted as \( T_i \), where \( i \) is the root of that tree. For each tree \( T_i \), proceed as follows: redefine the output of the adversary to be just the proof and the statement for the relation \( R_i^{up} \) (i.e., \( ((\text{crs}, c_i), \beta_i) \)), and the auxiliary input \( Z_i \) to be all the messages sent to the corrupt parties by Sim (on behalf of honest parties) prior to that, as well as the common reference string of \( \Pi_{vpe}(\rho) \); define the relation-tree with \( R_j^{up} \) for all \( j \) in \( T_i \), rooted at \( R_i^{up} \), and invoke the tree extractor for this relation tree and this adversary, with auxiliary input \( Z_i \). If the extraction succeeds, it yields witnesses for all the proofs \( \beta_j \) for all \( j \) in \( T_i \), and in particular, \( \hat{x}_j \) for all parties \( j \) in \( T_i \). If \( P_k \) is corrupt we define an adversary which outputs a statement \( (\text{crs}, c_k, \alpha) \) and a proof \( \gamma_k \) for the relation \( R_k^{down} \), where \( P_k \) is an (arbitrary) honest party adjacent to a leaf of \( T_i \). We define a relation-tree as follows: it consists of \( R_i^{down} \) for all \( j \) in the path from \( k \) to 1 (the root) and \( R_j^{up} \) for all \( j \in T_1 \) (except \( j = 1 \)); this tree is rooted at \( R_1^{down} \). Then we invoke the tree-extractor for the adversary with respect to this relation-tree, with auxiliary input \( Z_i \) which includes all the messages Sim sent to it till it produced the output. If successful, the extraction obtains the witnesses for all the proofs \( \gamma_j \) for all \( j \) in the path from parent(1) till the root, and for all the proofs \( \beta_j \) for all \( j \) in \( T_1 \) (except \( j = 1 \)). The witness for \( \gamma_j \) for \( j \in \text{chdrn}(1) \) includes \( \hat{x}_i \) and the witnesses for \( \beta_j \) include \( \hat{x}_j \) for all other \( j \) in \( T_1 \). If all the extractions are successful, Sim forwards the inputs \( \hat{x}_j \) for all corrupt parties \( P_j \) obtained above are forwarded to \( \mathcal{F}_{vpe}(\rho) \).

Execution Phase. Sim simulates the execution phase using the honest parties’ messages received from \( \mathcal{F}_{vpe}(\rho) \) at each round, by adding simulated proofs to them. Also Sim faithfully verifies all the signatures \( (c_i, \sigma_i) \) and proofs \( \delta_{ij} \), sent by each corrupt party \( P_i \) to any honest party \( P_j \), and if any verification fails, it sends (ABORT, \( j \)) to \( \mathcal{F}_{vpe}(\rho) \).

Figure 1: Simulator Sim used in the HYBRID_3 (ideal execution of \( \mathcal{F}_{vpe}(\rho) \)) for proving the security of \( \Pi_{vpe}(\rho) \).